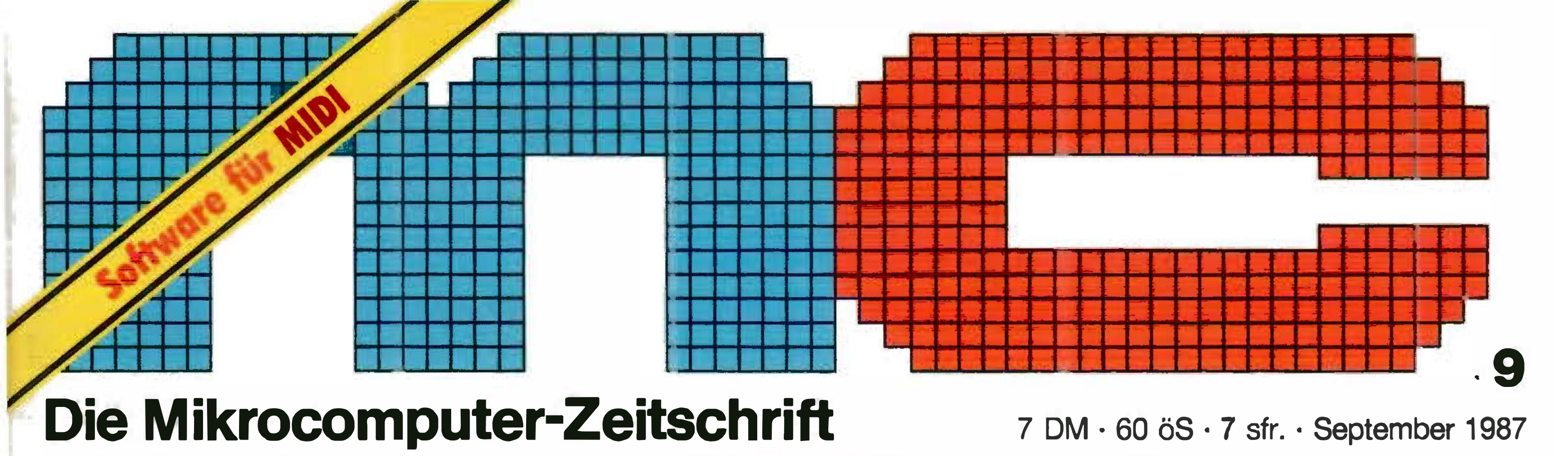


9

Die Mikrocomputer-Zeitschrift

7 DM · 60 öS · 7 sfr. · September 1987

PC-Technik:
Der mc-modular-AT

PC-Grafik:
Hercules-Hardcopy
EGA variabel

Grundlagen:
Neuronale Netzwerke

Atari ST:
GFA-Basic und Assembler
Eumel mit Elan

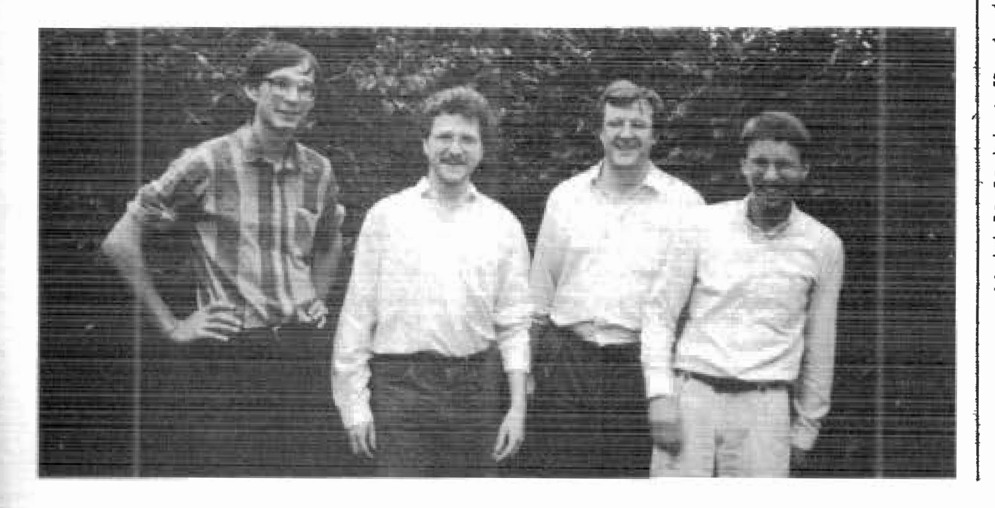
Macintosh:
Desktop Publishing auf SE



Liebe Leser!

Über dieses Heft ist mehr zu berichten, als hier eigentlich auf die Seite paßt. Beispielsweise über das in der unteren Bildreihe abgebildete Autorenteam. Und dazu gehört auch Wolfgang Back vom WDR-Computer-Club (rechts): mc bringt nämlich einen Beitrag zu der Sendung des WDR-Computer-Clubs im September. Er behandelt ein Thema, das alle Computeristen interessiert: wie dynamische Modelle von Nervennetzen natürliche Intelligenz erklären. Die Autoren kommen aus der Technischen Universität München; sie sind Mitglieder des Physik-Departements dieser Hochschule und forschen in der Arbeitsgruppe Theoretische Biophysik bei Professor Klaus Schulten. Wolfgang Back hat diese Arbeitsgruppe aufgesucht, weil sie sich mit Aspekten der künstlichen Intelligenz beschäftigt. Herausgekommen ist dabei der Beitrag „Physik und Gehirn". Einige faszinierende Ergebnisse werden in der Sendung gezeigt. Worauf wir stolz sind: Für mc haben die Autoren Pascal-Programme geschrieben, die (in aller Kürze) wesentliche Elemente der Arbeit zusammenfassen. Für Sie zum Experimentieren. Die Sendung wird in den dritten Programmen der Nordkette (WDR, NDR, SFB, Bremen und WDR) sowie im Hessischen Rundfunk zu unterschiedlichen Zeitpunkten ausgestrahlt. Bitte orientieren Sie sich in den Programmzeitschriften, da die Sendezeiten zum Zeitpunkt der Drucklegung diese Artikels noch nicht sicher feststehen. Ab 10. September.

Der Schwerpunkt in diesem Heft ist der mc-modular-AT – ein AT, wie Sie ihn haben wollen. Bei dieser Serie von mc kann jeder mitmachen, der verstehen will, wie ein AT funktioniert. Wir glauben nämlich, daß dieser fortschrittliche Standard-Computer jetzt für jeden anspruchsvollen Computermann transparent gemacht werden sollte. Einerseits, damit die Besitzer eines solchen Computers ihren Rechner besser verstehen, andererseits, damit die, die einen Computer erwerben wollen, wissen, was sie tun. mc-like dabei: Die Serie zeigt in übersichtlicher Form, welche Karten aus dem großen Angebot für Sie die richtigen sind. Mit mc einen modularen AT zusammenstellen – preiswerte Profi-Software gibt es ja in Hülle und Fülle.





Die Biber nagen an unseren Kräften. Bei der Überprüfung ist ein Punkt besonders schwierig: der Urlaub des einen oder anderen Biberzüchters. In Zweifelssfällen müssen wir zurückfragen, denn niemand soll nur aus Mangel an Information unsererseits ungerecht behandelt werden. Global kann schon gesagt werden: Ein Biber, der gewinnen soll, muß schon unter einer Sekunde 1915 Einsen produzieren. Wir haben Hardware-Biber bekommen und Software-Biber. Interessante Biber-Suchprogramme und viele Biber mit Einsen am Stück. Geduld, der Amiga 2000 ist noch nicht vergeben.

Ihr
Ulrich Rohde

mc-inhalt



Titelfoto: Klaus Hager

mc-modular-AT

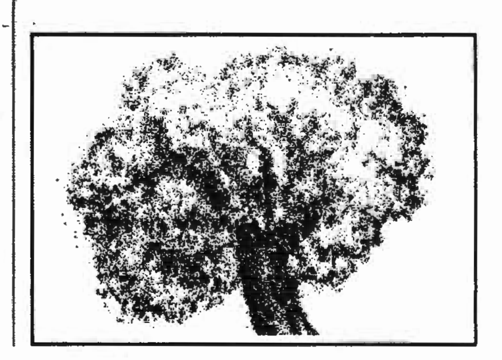
Mit dem mc-modular-AT stellen wir einen flexiblen Bausatz der Spitzenklasse vor. Alle Baugruppen von der CPU-Karte über den Grafik-Controller bis zum Floppy-Festplatten-Controller für 5¼- oder 3½-Zoll-Floppylaufwerke werden in der Artikelserie, die in dieser Ausgabe startet, ausführlich beschrieben.

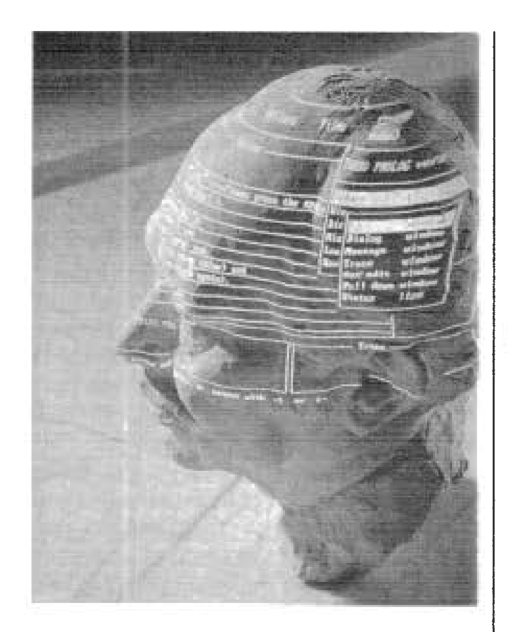
Seite 36

EGA-Spezial

EGA-Karten sind mit einem Bildspeicher von 256 KByte ausgestattet, der bei monochromem Betrieb reichlich überdimensioniert scheint. Mit einem Turbo-Pascal-Programm läßt sich die Auflösung nach Bedarf einstellen. Gleichzeitig stellen wir einen leistungsfähigen Grafik-Editor vor.

Seite 72



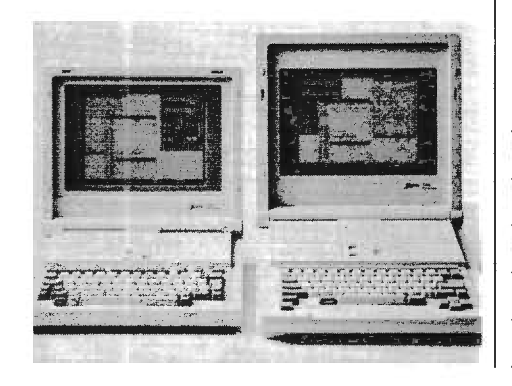


Neuronale Netze

Auf dem Weg zur künstlichen Intelligenz schauen die Forscher mehr und mehr bei der Natur ab. „Neuronale Netzwerke“ heißt das aktuelle Thema in der Grundlagenforschung, das wir mit praktischen Beispielen zum Mitmachen vorstellen. Wer sich verblüffen lassen möchte, liest den Artikel ab **Seite 108**

Tragbarer PC

Nur noch sechs Kilogramm muß man tragen, wenn man einen kompletten PC mit 20-MByte-Festplatte unterwegs dabei haben möchte. Neben dem schnellen Massenspeicher bietet der Neue von Zenith ein LC-Display in Supertwist-Technik für höchste Ansprüche. Unsere Meinung zu diesem Computer finden Sie ab **Seite 122**





Die Mikrocomputer-Zeitschrift

Franzis-Verlag GmbH
Karlstraße 37–41, 8000 München 2
Postfach 37 01 20, 8000 München 37
Telefon (0 89) 51 17-1
Telex 5 22 301
Telefax (0 89) 51 17-3 79
Tedas (0 89) 59 84 23 und 59 64 22

Chefredakteur:
Dipl.-Math. Ulrich Rohde

Stellvertretender Chefredakteur:
Dipl.-Ing. (FH) Ulrich Kruppe

Redaktion:
Redaktionsassistentin: Rita Schleser (0 89) 51 17-3 54
Ressortredakteure: Dipl.-Ing. (FH) Wolfgang Hascher (fl), Dipl.-Ing. (FH) Rudolf Hofer (fl), Reiner Schönrock, Dipl.-Ing. (FH) Dieter Strauß

Ständiger Mitarbeiter: Dipl.-Ing. (FH) Herwig Feichtinger (zu erreichen unter der Anschrift der Redaktion)

Korrespondent: USA: Stan Baker, 504 Nino Avenue, Los Gatos, CA 95030

Art Direktion: Dipl.-Designer (FH) Volker Hilbel

Layout, Grafik, Herstellung:
Josef Wurzinger

Franzis-Labor:
Neucom electronic GmbH

Software Service:
SHAMROCK Software Vertrieb (-3 31)

Verantwortlich für den Inhalt:
Dipl.-Math. Ulrich Rohde

Verantwortlich für Anzeigen:
Dietger Kötter

Gesamtherstellung:
Franzis-Druck GmbH
Karlstraße 35
8000 München 2
Telefon (0 89) 51 17-1

ISSN: 0720-4442

Vertriebskennzeichen:
B 7745 E



Sonderdrucke: Jakob Wintersberger

Lizenzen: Georg Geschke, Königstr. 8, 3000 Hannover 1, Telefon (05 11) 34 53 62

Monte Carlo

Zum Artikel „Monte Carlo im Computer" (mc 8/87) möchte ich eine kleine Ergänzung geben: Gerade bei der Modellierung physikalischer Vorgänge benötigt man häufig Paare (bivariable Verteilung) von stochastisch unabhängigen, normalverteilten Zufallszahlen (x_1, x_2), etwa bei der Simmulation des Standortfehlers elektronischer Navigationsverfahren, wo ein Standort üblicherweise aus zwei fehlerbehafteten Standlinien berechnet wird. Hier eine von M. D. Jöhnk 1967 veröffentlichte Lösung:

$$x_1 = \sqrt{-2 * \log(z_1)} * \sin(2\pi\ z_2)$$
$$x_2 = \sqrt{-2 * \log(z_1)} * \cos(2\pi\ z_2)$$

Dabei sind z_1 und z_2 gleichverteilte, unabhängige Zufallszahlen.
Mitunter benötigt man aber auch korrelierte (also nicht völlig voneinander unabhängige) Zufallszahlen (y_1, y_2), so z. B. wenn im obigen Fall zwei Standlinien aus nur drei Sendern abgeleitet werden. Dann erzeugt man zunächst wie oben unkorrelierte Zufallszahlen x_1, x_2 und transformiert dann

$$y_1 = x_1$$
$$y_2 = rho * x_1 + \sqrt{1 - rho^2} * x_2$$

wobei rho der gewünschte Korrelationskoeffizient ist.

Dipl.-Ing. K. E. Hewicker
5400 Koblenz

Eumel mit Elan

Laut gelacht habe ich über das Programmpaket Schulis mit dem Betriebssystem Eumel und der Programmiersprache Elan. Bei dem Gedanken an die Schüler bin ich dann allerdings auch etwas traurig geworden. Der Fall zeigt, daß die Schule ein geschlossenes System darstellt, das es nicht nötig hat, zu sehen, was in der Außenwelt gebräuchlich ist. Hauptsache die vermeintlichen eigenen Bedürfnisse sind erfüllt, den Schritt zu MS-DOS werden die Schüler später schon machen.
Die Methode ist vom Latein bekannt: Der Schüler bekommt eine für Unterrichtsbelange besonders geeignete Sprache (logisches Denken, sogenannte Bildung) serviert und kann daher später lebende Sprachen besonders leicht lernen – falls er dann noch Zeit und Kraft dazu hat. Wenn nicht, kann er immerhin mit Fremdwörtern und Zitaten imponieren.
Ich selbst habe 1970 die didaktisch hochwertige Programmiersprache Algol gelehrt bekommen, die sich kommerziell genausowenig durchgesetzt hat wie die Lehrsprache Pascal; normal wäre damals wohl Fortran gewesen und heute z. B. Basic oder C. Bemerkenswert ist, daß man bereits dabei ist, Eumel wenigstens an MS-DOS anzupassen, hoffentlich mit dem angebrachten Elan!

Dipl.-Ing. Heinz Geißendörfer
8500 Nürnberg

Frage I

Seit ich – seit kurzem – in Turbo-Pascal programmiere, stört mich die Aufblähung der .COM-Dateien durch die Laufzeit-Bibliothek. Ich war daher erfreut, als ich beim Durchsehen einiger mc-Hefte den Artikel „Sparprogramm" (mc 7/85 Seite 66) fand. Leider erfüllt das Programm meine Hoffnungen nicht: Es muß mehrere Fehler enthalten, die ich aber wegen mangelnder Erfahrung nicht finden kann. Können Sie bzw. der Autor des Artikels mir (und möglicherweise auch anderen Turbo-Anfängern) helfen?

Dr. Wolfgang Werbeck
7800 Freiburg

Antwort I

Das oben genannte Programm enthält keine Fehler, allerdings haben Sie das falsche Betriebssystem verwendet. Das abgedruckte Programm bezog sich auf CP/M und den Turbo-Pascal-Compiler zu diesem System. Bei der Verwendung unter MS-DOS müssen einige Änderungen vorgenommen werden. Das *Bild* zeigt eine geringfügig abgemagerte Version des Programmes für MS-DOS.

Günther Sternberg

```
      Line 1    Col 1   Insert    Indent  A:START.PAS
Program Start (Input, Output);

Var FD : File;                          (* File-Identifier *)

Begin
     If ParamCount = 0 then             (* kein Aufrufparameter vorhanden ? *)
       Begin
         WriteLn;
         WriteLn ('Fehler - Kein Aufrufparameter vorhanden !');
         WriteLn;
         Halt;
       End;
     Assign (FD, ParamStr (1) + '.CHN');  (* zu startende Datei vereinbaren *)
     (*$I-*) Chain (FD); (*$I+*)          (* Datei versuchen zu starten     *)
     If IOResult <> 0 then                (* Datei nicht vorhanden ?        *)
       Begin
         WriteLn;
         WriteLn ('Fehler - Datei nicht vorhanden oder I/O-Error !');
         WriteLn;
       End;
 End.
```

Vor dem Compilieren dieses Programmes muß im Options-Menü für die Code- und Daten-Segmentgröße der maximale Wert eingestellt werden, damit das zu startende Programm auch genug Platz hat

Frage 2

Ich möchte Sie bitten, meine Frage nach technischen Informationen über den IBM PCjr an Ihre Leser weiterzugeben: Es scheint – auch über IBM – unmöglich, über dieses in Europa nie verkaufte und inzwischen nicht mehr produzierte Gerät irgendwelche Informationen zu bekommen. Insbesondere benötige ich dringend einen Schaltplan, Informationen über Belegung der drei Steckplätze und der Schnittstellenbuchsen sowie Unterlagen über Speichererweiterung und Floppycontroller.

Ingo Siebler
4300 Essen 14

Frage 3

Ich habe früher Informatik studiert und bin jetzt Lehrer an einer Grundschule. Dadurch bin ich gesetzlich verpflichtet, bestimmte personenbezogene Daten meiner Schüler zu sammeln (z. B. Geburtsdatum, Konfession und Noten).
Während der letzten Jahre habe ich verschiedene Programme geschrieben, die mir das erleichtern. Bei Grundschulen sind schuleigene Computer noch nicht üblich, ich benutze also den eigenen PC (den das Finanzamt ohne Probleme anerkannt hat). In der Klasse hat es bisher noch keinen Einwand gegeben.
Natürlich ist ein Computer mit einer Klasse nicht ausgelastet. So sind mit der Zeit andere Dinge dazugekommen: Sportfestauswertung, Schulstatistik – auch hier werden personenbezogene Daten gespeichert. Ist dieses nun aber rechtens? Auf drei Anfragen (beim Schulträger, bei der Schulaufsicht und beim Datenschutzbeauftragten) erhielten wir drei gegensätzliche Auskünfte – von unbedenklich bis sofort löschen. Die Frage lautet also: Darf ich im eigenen Computer personenbezogene Daten speichern, die ich von berufswegen zu sammeln verpflichtet bin? Einholen des Einverständnises ist keine Lösung, denn dann müßten z. B. Statistiken oder Listen von Hand ergänzt und vor allem neu geschrieben werden. Damit wäre jeder Zeitvorteil verloren.

Ingo Küper
5840 Schwerte

Hydra...

Ich habe meine Computer mit der Hydra-3-Version programmiert, die ermittelten Zeiten möchte ich Ihnen nicht vorenthalten:

Sharp PC1401

Zahl	Zeit
9	3,4 s
99	4,2 s
999	7,8 s
9999	10,4 s
99999	35,8 s

Acorn B (BBC)

Zahl	Zeit
9	3,6 s
99	2,94 s
999	4,47 s
9999	7,23 s
99999	6,86 s
999999	6,27 s
9999999	8,25 s

Mit der Hydra Version 3 benötigt der BBC für die Zahlen von 1...1000 genau 17 Minuten und 24,53 Sekunden.

Rainer Korfin
2000 Hamburg 74

WordStar-Patch

Ich wende mich an Sie wegen eines speziellen Anpassungsproblemes des Textverarbeitungsprogrammes WordStar 3.0 an einen CP/M-2.2-Rechner. Kurz meine Wunschvorstellung: Ich möchte WordStar direkt aus dem Betriebssystem heraus, im Mailmerge-Modus mit Mixdruck beginnend, aufrufen. Es soll eine Steuerdatei, zum Beispiel mit dem Namen „DRUCK" erstellt werden, die alle Punktkommandos für Mixdruck enthält. Danach möcht ich WordStar vom Betriebssystem her durch eine Sub-Datei aufrufen, zum Beispiel durch „WSDRUCK", und der gewünschte Mixdruck soll abgearbeitet werden. Sind Ihnen entsprechende Patchadressen bekannt – oder besteht die Möglichkeit bei den mc-Lesern nachzufragen?

Bernd Ciliox,
6340 Dillenburg 2

Änderung von „HerculesWrite"

Im Listing des Artikels „Turbo-Pascal bedient Hercules-Karte" (mc 2/87) erscheint unter anderem die Prozedur „HerculesWrite". Diese kann Characters über ASCII (127) nur dann ausgeben, wenn eine eigene Zeichentabelle eingerichtet wurde.

Folgende Lösung bietet sich an:

1. Laden des DOS-Programmes „Graphtabl.Com".

2. Ermittlung der Adresse der soeben geladenen Zeichentabelle mittels INT-21 (AH auf $35 setzen AL auf $1F setzen Resultat in ES:BX).

3. Benötigtes Zeichen aus der Tabelle lesen.

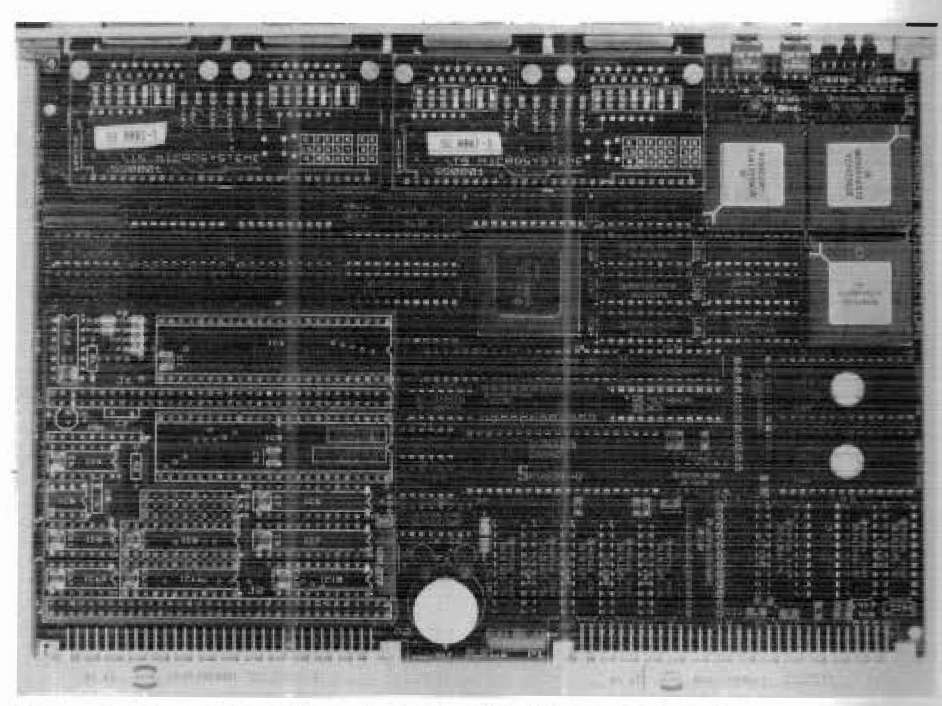
Joachim Simon
A-8010 Graz

```
 procedure hercules_write ( x,y    : integer;
                            actchar : char);
type
  regs       = record
               ax,bx,cx,dx,bp,si,di,ds,es,flags : integer;
               end;
 var
   r                 : regs
   i, offs ,charoffs : integer;
   txtbl             : array [0..127,0..7] of byte absolute $ffa6:$e;

 begin
   charoffs   := ord (actchar);
   if charoffs < 128 then
   begin
     for i  := 0 to 7 do
     begin
       offs := sys_byteoffset (x,y+1);
       mem [$b800:offs] := txtbl [charoffs,i];
     end;
   end
   else
   begin
     r.ax := $351f;
     msdos (r);
     for i := 0 to 7 do
     begin
      offs := sys_byteoffset (x,y+1);
      mem [$b800:offs] := mem [r.es:((charoffs - 128) * 8 + i + r.bx)];

     end;
   end;
 end;
```

Nun funktioniert HerculesWrite



Einen kompletten Unix-Computer stellt diese Platine von Thomson dar

Unix auf einer Platine

Auf einer einzigen Platine hat die Firma Thomson einen kompletten Unix-Computer untergebracht. Die CPU-Karte TSVME-106 basiert auf dem Prozessor 68010, der mit 10 MHz betrieben wird. Die Platine ist mit einer Speicherverwaltungseinheit (68451), einem Gleitkomma-Coprozessor (68881) und einem DMA-Controller (68440) bestückt. Außerdem enthält die Platine bis zu 4 MByte Dual-Port-RAM und 128 KByte EPROM mit einem Selbsttest-Debugger-Monitor-Programm.

Neben vier seriellen Schnittstellen bietet die Platine ein Centronics-Interface und einen SCSI-Adapter mit einer Übertragungsrate von 1,5 MByte/s. Außerdem stehen ein CMOS-Kalendermodul mit Batteriepufferung und eine programmierbare Echtzeituhr zur Verfügung.

Komplettes Desktop-Publishing-System von Osborne

Osborne ist seit kurzem Generallizenznehmer der amerikanischen Firma Advanced Vision Research (AVR). Damit kann das Unternehmen jetzt ein komplettes Desktop-Publishing-System, bestehend aus Hardware und Sofware, anbieten. Neben einem PC-kompatiblen Computer enthält das Grundpaket entweder den „Pagemaster-1" oder den „Pagemaster-J". Das erste Produkt besteht aus dem Softwarepaket „MegaScan" und der Einsteckkarte „MegaBuffer"; zum zweiten gehört zusätzlich der Flachbett-Scanner AVR-300.



Alles, was man zum Desktop-Publishing braucht, gibt es jetzt von Osborne

Die Software dient als Schnittstelle zwischen Scanner, Einsteckkarte (mit Pufferspeicher) und Laserdrucker. Sie erlaubt die Bild- und Text-Editierung und kann per Tastatur oder Maus bedient werden. Die GEM-Benutzerschnittstelle erleichtert die Arbeit mit diesem Programm.

Auf der MegaBuffer-Karte befindet sich 1,3 MByte RAM, das nicht in den Systemspeicher des PC eingebunden ist. Scanner oder Drucker können diesen Bereich nutzen. Die Kapazität reicht aus, um ein DIN-A4-Dokument mit einer Auflösung von 300 Punkten/Zoll aufzunehmen. Erweiterungen dieses Adreßraums sind problemlos möglich.

Der Flachbett-Scanner AVR-300 digitalisiert Vorlagen in Form von losen Blättern oder Büchern mit einer Größe von maximal 8,5 Zoll × 11,7 Zoll und einer Auflösung von 300 Punkten/Zoll. Graustufen ergeben sich aus der unterschiedlichen Punktdichte. Für ein Bild braucht das Gerät neun Sekunden.

Datenübertragung per TAXI

Die Welt der Elektronik ist um eine wohlklingende Abkürzung reicher: TAXI nennt die Firma AMD ihre neue Schnittstelle, die mit vollem Namen „Transparent Asynchronous Transmitter/Receiver Interface" heißt. Was dahintersteckt, ist bemerkenswert: Zwei neue Chips, der TAXI-Transmitter Am7968 und der TAXI-Receiver Am7969, sorgen für die Datenübertragung auf Koaxialkabeln oder Lichtleitern mit einer Übertragungsgeschwindigkeit von 56 MBit/s. Noch in diesem Jahr will AMD die Geschwindigkeit bis zu 100 MBit/s steigern. Im Vergleich zu heute üblichen seriellen Verbindungen mit Übertragungsraten um die 10 MHz kann sich das neue Konzept wahrlich sehen lassen.

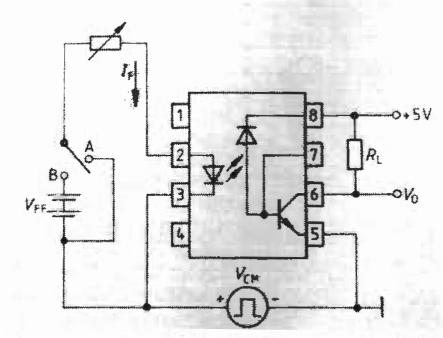
Dabei muß sich der Anwender keineswegs mit schwierigen „Protokollfragen" herumschlagen. Transmitter und Receiver werden einfach über ihre Schnittstellenleitungen an Busse mit maximal 10 Bit Breite angeschlossen. Für breitere Busse kaskadiert man die Bausteine. Um die Umwandlung in den seriellen Bitstrom und zurück in das parallele Format kümmern sich die Chips selbst.

Commodore: Ein Drittel des Umsatzes in Deutschland

Commodore macht wieder Gewinn: Auch im dritten Quartal des laufenden Geschäftsjahrs schrieb das Unternehmen schwarze Zahlen. Bei einem Umsatz von 169,5 Mio. $ betrug der Gewinn 1 Mio. $. Noch im entsprechenden Vorjahreszeitraum hatte das Unternehmen einen Verlust von 36,7 Mio. $ gemacht. Damit schreibt man jetzt zum viertenmal hintereinander eine positive Quartalsbilanz. Insgesamt ergibt sich für die drei zurückliegenden Quartale eine Gewinn von 26,5 Mio. $ bei einem Gesamtumsatz von 616,3 Mio. $.
Etwa 70% des Gesamtumsatzes macht Commodore außerhalb der USA. Den größten Anteil davon, nämlich rund die Hälfte, erwirtschaftet die deutsche GmbH. Damit trägt sie zu einem Drittel zum Gesamtumsatz des Unternehmens bei. Im dritten Quartal des laufenden Geschäftsjahres betrug der Umsatz der deutschen GmbH 107,6 Mio. DM, was einer Steigerung von rund 20% gegenüber dem gleichen Vorjahreszeitraum entspricht.
Die Verkaufszahlen der hiesigen Vertriebsgesellschaft sind beeindruckend: Rund 177000 Rechner aller Kategorien wurden umgesetzt, davon über 152000 Heimcomputer, rund 20000 PCs und etwa 5000 Amigas.

Optokoppler schafft 1 MBit/s

Siemens hat einen Optokoppler auf den Markt gebracht, dessen Fotoempfänger in eine Diode und einen Transistor aufgetrennt ist. Diese „Arbeitsteilung" sorgt dafür, daß der lichtempfindliche Diodenteil der Empfängerschaltung nur noch um den Betrag der Signalspannung (wenige Millivolt) umgeladen werden muß. Bei konventionellen Kopplern ist die Umladezeit bzw. die Arbeitsgeschwindigkeit um eine Größenordnung ungünstiger. Der Optokoppler (6N135/136) kann bis 1 MBit/s eingesetzt werden. Er ist zehnmal schneller als konventionelle Transistorkoppler.



Schneller als herkömmliche Typen ist dieser Optokoppler von Siemens

Seminare

Das Haus der Technik e. V. in Essen führt gemeinsam mit dem Institut für Organisationsforschung und Technologieanwendung (IOT), München, den Kurs „Optimale Nutzung von Bürokommunikationssystemen" durch. Termine: Donnerstag 17. September und Freitag 18. September 1987. Experten des IOT entwickeln dabei mit den Teilnehmern eine Strategie, die den büro- und verwaltungstypischen „Leidensdruck" abbauen hilft.

Eine Reihe von Seminaren bietet die Dortmunder Firma OSS zum Thema „Optische Speichersysteme" an. Die Seminare finden im September und im Oktober in Hannover, Düsseldorf, Frankfurt, Ettlingen und München statt.

Die Racal-Milgo GmbH, Neu-Isenburg, führt in diesem Jahr noch mehrere Seminare zu den Themen „Datenkommunikation" und „X.25 – Datenpaketvermittlung und Datex-P" durch. Sie richten sich an Techniker mit einschlägigen Vorkenntnissen.

Im hauseigenen Schulungszentrum bietet die Münchner Firma PCS im September und Oktober Kurse zu den Themen „Unix" und „C" an.

Im Auftrag des Bundesministeriums für Bildung und Wissenschaft führt die Wirtschaftsakademie für Lehrer e.V. in Bad Harzburg einen Modellversuch durch, bei dem arbeitslose Lehramtskandidaten zu Technologieassistenten umgeschult werden. Die Teilnehmer absolvieren ein einjähriges Fernstudium mit begleitendem Direktunterricht in Form von Kurzseminaren und Arbeitsgemeinschaften. Ferner werden 60tägige Computerseminare durchgeführt. Auskünfte gibt es unter der Telefonnummer 0 53 22/7 30 oder 7 33 10.

Motorola führt unter anderem wieder Seminare über das Betriebssystem „Unix" (13.–16. Oktober in München), den VME-bus (28.–29. Oktober in Düsseldorf) und die Programmiersprache „C" (9.–12. November in München) durch.

„Entwickeln von Expertensystemen" heißt ein eintägiger Workshop, der den Einstieg in dieses Gebiet vermitteln will. Veranstalter ist die Firma ExperTeam GmbH, München. Veranstaltungsorte sind Köln (24. September) und München (14. September und 5. Oktober). Für die Teilnahme sind keine Vorkenntnisse erforderlich.

Sicherheitsaspekte in Datennetzen und rechtliche Fragen im Netzbetrieb werden am 1. und 2. Oktober 1987 in Köln auf einem Symposium der Gesellschaft für Datenschutz und Datensicherung e.V., Bonn, diskutiert. Anmelden kann man sich unter Tel. 02 28/69 43 31.

Vom 30. September bis zum 2. Oktober 1987 wird an der Universität Karlsruhe die Arbeitstagung „Rechnerarithmetik, Wissenschaftliches Rechnen und Programmiersprachen" veranstaltet. Das umfangreiche Tagungsprogramm kann beim Institut für Angewandte Mathematik in 7500 Karlsruhe 1, Tel. 07 21/6 08 26 79 angefordert werden.

Termine

Um alle Aspekte der Datensicherheit geht es bei der „Compsec 87 – Fourth National Computer Security Conference", die am 28. und 29. Oktober in London stattfindet.

Am 7. und 8. November findet auf dem Gelände der Hannover-Messe die „Interradio '87" statt. Die Veranstaltung trägt den Untertitel „6. Internationale Ausstellung für Amateurfunk, Hobbyelektronik und Computertechnik".

Die „Erste Internationale Konferenz für Telekommunikation" findet am 12. und 13. November in Amsterdam statt. Die veranstaltende Firma, PA Computer, Frankfurt, will damit einen Beitrag zur Entwirrung der derzeit unübersichtlichen Lage auf dem weltweiten Telekommunikations-Sektor leisten.

„IMPRINTA – Internationaler Kongreß und Ausstellung für Kommunikationstechnik" nennt sich eine Veranstaltung, die die Wechselbeziehung zwischen Büro-, Druck- und Telekommunikationstechnik aufzeigen will. Sie findet vom 18. bis 24. Februar 1988 in Düsseldorf statt.

Europa bei der Datenverarbeitung sparsam

In seiner Studie „Perspektiven '87" vergleicht das Eschborner Marktforschungsinstitut IDC die Ausgaben der Industrienationen für die Datenverarbeitung. Das Ergebnis der Untersuchung bezeichnet Günter Bröking, Marketing- und Vertriebsleiter bei IDC, als erschreckend für jeden Europäer. Die nebenstehende Aufstellung zeigt, aufgrund welcher Zahlen er zu dieser Beurteilung kommt.

USA	5,4 %
JAPAN	6,0 %
WESTEUROPA	2,3 %
Deutschland	2,1 %
Frankreich	2,3 %
England	2,7 %
Italien	2,2 %
Holland	2,2 %
Spanien	1,4 %
Schweden	2,4 %
Schweiz	2,9 %
Belgien	2,0 %
Dänemark	2,4 %
Österreich	2,2 %
Norwegen	2,6 %
Finnland	2,4 %

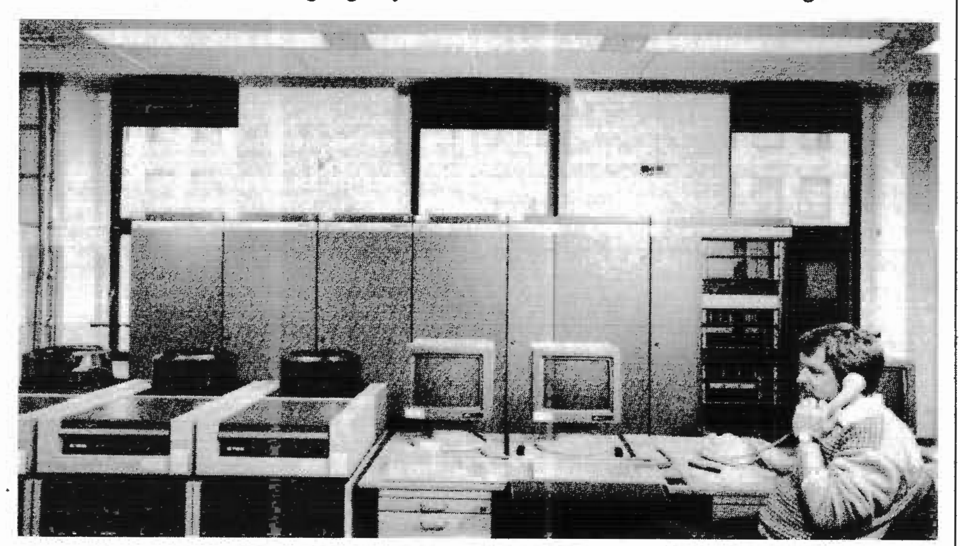
Anteil der Ausgaben für die Datenverarbeitung vom Bruttosozialprodukt (nach einer IDC-Studie)

Während Japan im letzten Jahr 6 % des Bruttosozialproduktes in die Datenverarbeitung investiert hat, liegt der Anteil der DV-Ausgaben vom Bruttosozialprodukt in den europäischen Ländern durchschnittlich bei 2,3 %. Noch unterhalb des europäischen Durchschnitts liegt die Bundesrepublik Deutschland mit 2,1 %. Die USA erreichen mit 5,4 % immerhin einen Wert, der sich annähernd mit dem der Japaner messen kann.

Nach Einschätzung von Günter Bröking wird diese Situation für Westeuropa und speziell für die Bundesrepublik Deutschland zur Gefahr werden. Je mehr die Informationstechnik zur strategischen Waffe im Wettbewerb der Zukunft werde, desto deutlicher müßten auf Dauer europäische Unternehmen denen der USA und Japans unterlegen sein, meint Bröking.

AUTEX gibt Auskunft

Anfang Juni nahm die Deutsche Bundespost das automatische Telex-Teletex-Auskunftssystem AUTEX in Betrieb. Die vorhergehende Erprobungsphase (seit Ende 1986) hatte durchweg positive Erfahrungen mit dem in Darmstadt stehenden Rechner ergeben. AUTEX ersetzt das seit 1973 bestehende Vorgängersystem: die automatische Telexauskunft. Neu ist, daß jetzt erstmalig ein weltweit nutzbares automatisches Auskunftssystem auch für den Teletex-Dienst zur Verfügung steht. Im Durchschnitt werden täglich 6900 Anfragen gestellt, von denen rund 60 % innerhalb weniger Sekunden automatisch beantwortet werden. Dazu gehören Anfragen über die auf Datenträger gespeicherten Teilnehmer der Bundesrepublik, der DDR, der Schweiz und Liechtensteins. Das System soll in naher Zukunft um weitere ausländische Netze erweitert werden. Auskünfte über Teilnehmer aus nicht gespeicherten Ländern und allgemeine Anfragen beantworten nach wie vor Mitarbeiter der Bundespost.



Rund 170 000 Telex- und 16 000 Teletex-Teilnehmern steht das automatische Auskunftssystem AUTEX kostenlos zur Verfügung

Neue ELO-Sonderhefte

„Elektronik für Einsteiger" heißt ein neues Sonderheft unserer Schwesterzeitschrift ELO. Es liefert das wichtigste Grundwissen, das man braucht, wenn man in das Hobby Elektronik einsteigen will. Die Beiträge reichen vom allgemeinen elektronischen Rüstzeug über Anregungen für die praktische Arbeit (Schaltungen auflösen, Löten, Platinen ätzen) bis hin zur Computertechnik. Das 96 Seiten starke Heft kostet 10 DM.

Ein zweites Sonderheft der ELO wendet sich an den fortgeschrittenen Elektronikbastler. Es trägt den Titel „Das interessante IC" und erweist sich als wahre Fundgrube für den Hobbyisten. Von fast 40 integrierten Schaltungen sind Anschlußbelegung, Blockdiagramm, Schaltungsbeispiele und eine leicht verständliche Erklärung abgedruckt. Das Heft ist 80 Seiten stark und kostet 12,80 DM.

Die Sonderhefte sind bei allen Bahnhofsbuchhandlungen, beim Elektronik-Fachhandel, in Mikrocomputer-Shops, bei größeren Zeitschriftenverkaufsstellen, in Buchhandlungen und direkt beim Franzis-Verlag erhältlich.

Logitech-Maus für IBM PS/2-Systeme

Speziell für die neuen Modelle der IBM PS/2-Serie hat Logitech die Logitech Series/2-Maus entwickelt, die direkt an das Maus-Port angeschlossen wird und voll kompatibel zu den neuen IBM-Modellen ist. Nach Angaben des Unternehmens zeichnet sich das Eingabegerät durch eine bessere Qualität und ein ansprechenderes Design dem Original gegenüber aus. Mit der Plus-Package-Software steht dem Anwender neben dem eigentlichen Maus-Treiber ein System zum Aufbau von Pop-Up-Menüs, eine Schnittstelle zu Lotus 1-2-3 und ein leistungsfähiger Texteditor zur Verfügung.

Vicki kommt mit verbesserter Software

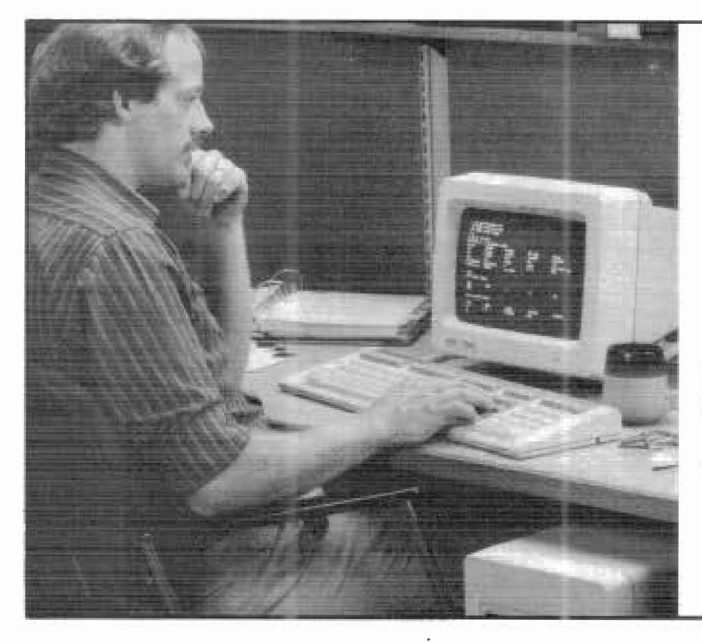
Als weitere Firma „mit Namen“ kommt jetzt Victor mit einem sehr preiswerten PC auf den Markt. In der Einstiegsversion kostet Vicki rund 1700 DM. Für diesen Preis bekommt man den PC mit Tastatur, 12-Zoll-Monitor, einem 5¼-Zoll-Disketten-Laufwerk, drei langen Steckplätzen, einer Parallel-Schnittstelle und eingebautem Anschluß für Lichtgriffel oder Maus. Bereits auf der Systemplatine ist ein Hercules- und CGA-kompatibler Grafikadapter integriert. In der Grundausführung ist das Gerät mit 512 KByte RAM bestückt. Größere Versionen mit einem weiteren Disketten-Laufwerk oder einer Harddisk sowie mit Echtzeituhr und 14-Zoll-Monitor gibt es ebenfalls. Als CPU verwendet Victor den Prozessor 8088, der sich zwischen Taktfrequenzen von 4,77 und 7,16 MHz umschalten läßt.
Als Betriebssystem wird MS-DOS 3.2 eingesetzt, das die Firma um Hilfsprogramme aus dem Unix-Bereich ergänzt hat. Anstelle von Standard-GWBasic erhält der Käufer mit VBasicA eine verbesserte Version.



Wieder ein PC-Kompatibler: Victors Modell Vicki

System-320 meistert Echtzeit-probleme

Insgesamt vier Modelle für den Mehrbenutzerbetrieb enthält Intels neue Computerfamilie „System-320“. Sie basieren alle auf dem 32-Bit-Prozessor 80386 und kommen auf eine Verarbeitungsleistung von maximal 4 MIPS (Mio. Befehle pro Sekunde). Für diesen beeindruckenden Wert ist unter anderem ein 64-KByte-Cache verantwortlich.
Die neuen Computer laufen einerseits unter Xenix, andererseits unter dem Echtzeit-Betriebssystem iRMX-286. Von letzterem hat Intel jetzt eine neue Version (Release 2) herausgebracht. Sie bietet Vorteile in bezug auf Benutzerumgebung und Netzwerkfähigkeit. So ist mit dem High-Level-Language-Debugger Softscope ein komfortables Werkzeug vorhanden, das die Fehlersuche auf Hochsprachenebene erlaubt.



Zusammen mit einer neuen iRMX-286-Version wurde Intels neue Computerfamilie angekündigt

Image-Scanner im Vormarsch

Sogenannte Image-Scanner zum Erfassen von Bildern werden künftig mehr und mehr an Bedeutung gewinnen. Zu diesem Schluß kommt das Eschborner Marktforschungsinstitut International Data Corporation (IDC) in seiner jüngsten Studie „Office Image Scanner Market 1985-1990“. Im technisch-wissenschaftlichen Bereich sind Image-Scanner schon jahrzehntelang im Einsatz. Ein Beispiel dafür sind CAD/CAM-Anwendungen. Die Preise für solche Geräte erstrecken sich von 50 000 bis zu 1 Mio. DM.
Derzeit drängen immer mehr Scanner für den PC-Bereich auf den Markt. Sie werden schon zu Preisen von unter 10 000 DM angeboten. Die Studie weist rund 20 Unternehmen aus, die Scanner dieser Klasse anbieten, darunter Canon, Datacopy, Dest, IBM, MicroTek, Panasonic, Ricoh, Tecmar, Wang und andere. IDC sagt voraus, daß zum Ende dieses Jahres die meisten Anbieter von Büroinformationstechnik PC-Image-Scanner im Angebot haben werden.

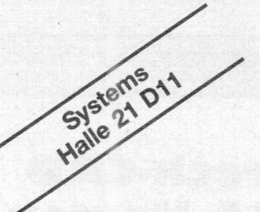
Apple wieder erfolgreich

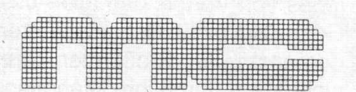
Im dritten Quartal des Geschäftsjahres 1986/87 hat die Firma Apple einen Umsatz von 637,1 Mio. $ erzielt. Das entspricht einer Steigerung von 42% gegenüber dem gleichen Vorjahreszeitraum (448,3 Mio. $). Der Gewinn stieg um 65% auf 53,5 Mio. $. Apple-Chef John Sculley führt diese erfreulichen Zahlen zum Teil darauf zurück, daß die neuen Produkte (z. B. der Macintosh II) im Markt großen Anklang fänden.

Intel schreibt Rekordzahlen

Mit 349 Mio. $ konnte die Intel Corporation im zweiten Quartal dieses Jahres einen Rekordumsatz verzeichnen. Der Nettogewinn betrug 46 Mio. $. Noch im gleichen Zeitraum des Vorjahres hatte die Firma einen Verlust von 20 Mio. $ hinnehmen müssen (bei 305 Mio. $ Umsatz). Nach den Worten des Firmenpräsidenten Andrew S. Grove ist diese Entwicklung unter anderem auf die große Nachfrage nach den Prozessoren 80286 und 80386 zurückzuführen.

GRAF® computer





- modular - AT

Völlig AT-kompatibel. Takt: 10/12 MHz. Arbeitsspeicher 512K, auf 1 MByte ausbaubar. Ausrüstbar mit Floppy-Laufwerken: IBM PC, IBM AT und IBM Personal System/2-kompatibel. Festplatte 20 MByte (+ 20 MByte). Modulares, ausbaufähiges System.

Nr.	Best.-Nr.	Beschreibung	Preis DM*
–	11076	**Grundversion** des mc-modular AT, enthält die folgenden Einzelteile, die auch **einzeln** erhältlich sind:	2798,–
1	10961	**CPU-Baugruppe**, enthält alle Elemente eines AT-Motherboards	1198,–
2	11066	**Festplatten-Floppy-Controller**, steuert 1 ... 4 Disk und 1 ... 2 Festplatten-Laufwerke **auch 3½" – PS/2!**	398,–
3	11075	**Herkules-Graphik / Druckeradapter**	148,–
4	10955	Passiver **Bus**, 8 Einbauplätze (6 lange)	98,–
5	11073	TEAC-**Laufwerk** FD-55GFR 5¼" / 1.2 MB	299,–
6	11017	**Gehäuse**, mit allen Bohrungen	159,-
7	11010	MF-2-kompatible **Tastatur**	298,–
8	11039	**Netzteil**, 190 W, ausreichend für Vollausbau	198,–
–	10963	Befestigungs- und **Kabelsatz** (1 x Floppy, 1 x Platte)	59,–
–	11074	**Handbuch**	30,–

Nr.	Best.-Nr.	Beschreibung	Preis DM*
–	11077	**mc-modular-AT-Komplettversion**, enthält die oben angeführten Bauteile der Grundversion **und zusätzlich:**	3798,–
9	11013	**Monitor**, amber	325,–
10	10964	**Festplatte**, Seagate, ST 225, 20 MByte, 56 ms	698,–
–	11078	**mc-modular-AT-Fertiggerät**, komplett **1 Jahr Garantie.** Enthält alle Bauteile der oben angeführten Fertigversion, fertig aufgebaut, geprüft sowie zusätzlich:	4498,–
–	10960	**MS-DOS 3.1** Betriebssystem	198,–

Weitere Baugruppen, wie serielle Schnittstellen, EGA-Karten, 3½"-Floppy-Laufwerke etc. auf Anfrage!

* Die Preise in dieser Anzeige sind auf Grund des Anzeigenschlusses mindestens vier Wochen alt und können sich sehr schnell ändern. Fragen Sie bitte die **aktuellen** Preise telefonisch an. Alle Preise verstehen sich freibleibend, ab Kempten, und **beinhalten** die ges. Mehrwertsteuer. Die Produkte sind auch in unseren Filialen erhältlich.

Bestellung oder Anfrage mit anhängender Karte

GRAF ELEKTRONIK SYSTEME GMBH

Magnusstraße 13 · Postfach 1610 · 8960 Kempten (Allgäu) · Telefon: (0831) 6211
Teletex: 831804=GRAF · Telex: 17831804=GRAF · Datentelefon: (0831) 69330

Shamrock-CAD für EGA-Karten

Ab sofort ist Shamrock-CAD in einer Version für EGA-Karten lieferbar. Die EGA-Karte wird von diesem Programm im monochromen Modus (Auflösung von 640 × 350 Punkte) betrieben. Um das Aufsetzen des Cursors auf die Rasterpunkte zu erleichtern, wurde ein neuer Programm-Modus hinzugefügt: der Snap-Modus. Damit kann der Anwender Zeichnungen wesentlich schneller erstellen.

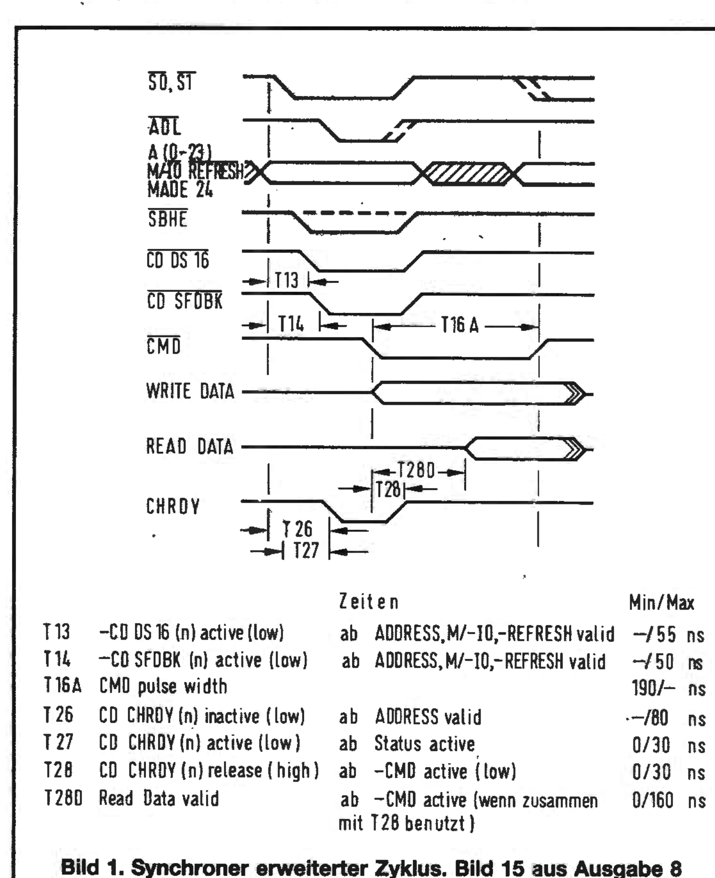
Einladung erwünscht

Unter vielen Computerunternehmen macht sich wachsender Verdruß über große Computermessen wie die CeBIT in Hannover oder die Systems in München breit. Diese beanspruchen nicht nur einen immensen Aufwand an Zeit- und Personal, sondern häufig wird auch nicht der Besucherkreis erreicht, der letztendlich durch die Aussteller angesprochen werden soll.
Beverley Johnson, eine amerikanische Marketingexpertin aus dem Elektronik- und Computerbereich, entdeckte hier eine Marktnische und gründete 1972 ein Marketingunternehmen, das sich voll auf die Organisation sogenannter „Invitational Computer Conferences" (ICCs) spezialisiert hat.
ICCs sind eintägige regionale Ausstellungen für Computer und deren Zubehör auf denen zusätzlich noch Fachseminare abgehalten werden. Sowohl die Aussteller als auch die Besucher werden vom Veranstalter eingeladen. Sinn und Zweck der auf verschiedene Zielgruppen abgestellten Ausstellungs- und Seminarprogramme besteht darin, Computer-Fachleute über neue Markt-Trends und Produkt-Technologien zu informieren.
Zu den bereits seit mehreren Jahren veranstalteten OEM-Peripherie-Konferenzen sind jetzt die PC-Wiederverkäufer- und die Computergrafik-Konferenzen hinzugekommen. Besonders für kleinere Fimen ist es von Vorteil, daß alle Aussteller ihre Geräte auf einheitlich gestalteten Ständen demonstrieren. Großfirmen können daher nicht allein durch ihre Standfläche kleinere Firmen quasi erdrücken. Im Vordergrund der ICCs steht die Technologie und nicht die Werbung.
Allein schon im Interesse der auf den ICCs nicht zugelassenen Endanwender bleibt allerdings zu hoffen, daß dieses Marketingkonzept nicht die großen Messen überflüssig macht. Denn wo sollte sich der Endanwender oder der einfach nur an Computern Interessierte noch umfassend informieren?

Nachtrag

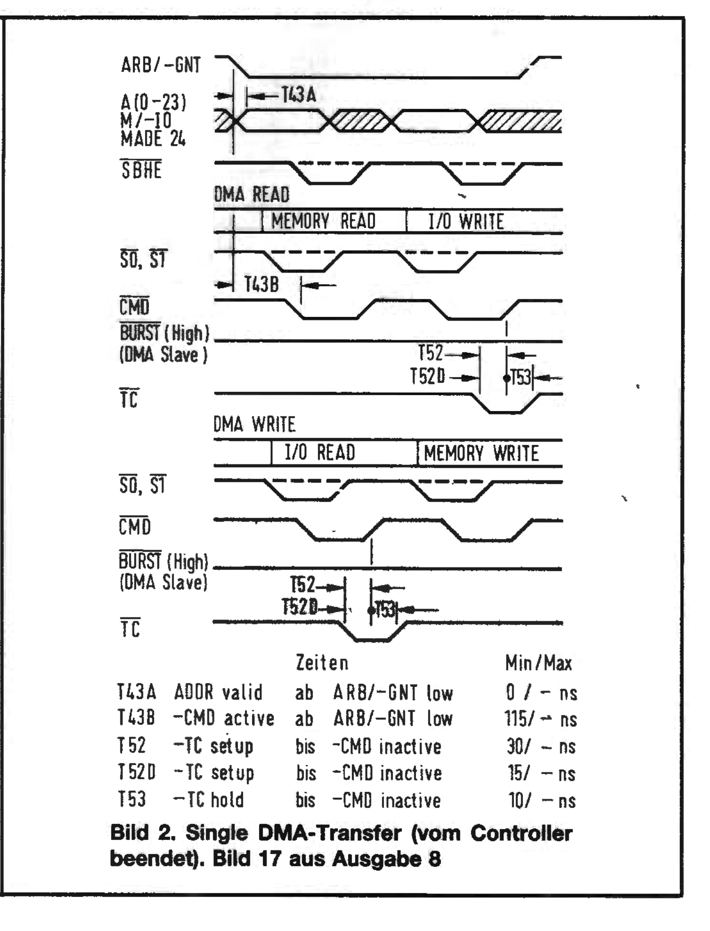
Fehlerteufel im Mikro-Kanal

Im Artikel zum IBM-Mikro-Kanal-Bus (mc 8/87) wurden Bilder teilweise vertauscht und teilweise doppelt reproduziert, wobei das richtige Bild dann fehlte. Auf Seite 45 der Ausgabe 8/87 müssen die Bildunterschriften von Bild 10 und 11 vertauscht werden.
Auf der Seite 47 ist anstelle von Bild 15 nochmals das Bild 16 abgebildet. Das richtige Bild 15 sei hier als *Bild 1* nachgeliefert.
Bild 2 in diesem Nachtrag zeigt das richtige Bild 17 auf Seite 48 des Beitrages. Dort war der Inhalt von Bild 18 ebenfalls versehentlich in Bild 17 geraten. Wir bitten um Vergebung.



		Zeiten	Min/Max	
T13	-CD DS16 (n) active (low)	ab ADDRESS,M/-IO,-REFRESH valid	-/55	ns
T14	-CD SFDBK (n) active (low)	ab ADDRESS,M/-IO,-REFRESH valid	-/50	ns
T16A	CMD pulse width		190/-	ns
T26	CD CHRDY (n) inactive (low)	ab ADDRESS valid	-/80	ns
T27	CD CHRDY (n) active (low)	ab Status active	0/30	ns
T28	CD CHRDY (n) release (high)	ab -CMD active (low)	0/30	ns
T28D	Read Data valid	ab -CMD active (wenn zusammen mit T28 benutzt)	0/160	ns

Bild 1. Synchroner erweiterter Zyklus. Bild 15 aus Ausgabe 8



		Zeiten	Min/Max
T43A	ADDR valid	ab ARB/-GNT low	0 / - ns
T43B	-CMD active	ab ARB/-GNT low	115/ - ns
T52	-TC setup	bis -CMD inactive	30/ - ns
T52D	-TC setup	bis -CMD inactive	15/ - ns
T53	-TC hold	bis -CMD inactive	10/ - ns

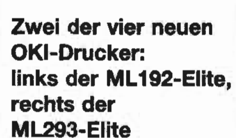
Bild 2. Single DMA-Transfer (vom Controller beendet). Bild 17 aus Ausgabe 8

Microline-Familie bekommt Zuwachs

Vier neue Varianten der erfolgreichen Microline-Drucker-Familie hat die Firma OKI vorgestellt. Gegenüber den Vorgängertypen verfügen die Modelle ML192/193-Elite und ML292/293-Elite über zusätzliche Druckgeschwindigkeiten: Super-Schnelldruck und Schnelldruck.

Im Super-Schnelldruck erreichen die beiden kleineren Modelle 240 Zeichen/s, die beiden größeren bringen es dabei auf 300 Zeichen/s. Die entsprechenden Werte im Schnelldruck sind 200 und 240 Zeichen/s. Alle Geräte beherrschen auch „Near Letter Quality", dabei erreichen sie Geschwindigkeiten von 40 bzw. 100 Zeichen/s.

Die vier neuen Modelle sind von Haus aus Epson- und IBM-kompatibel. Ihr Puffer ist einheitlich 16 KByte groß.



Zwei der vier neuen OKI-Drucker: links der ML192-Elite, rechts der ML293-Elite



Neue Modellreihe von Olivetti

Mehrere neue PC-Modellreihen hat die Firma Olivetti angekündigt. Es handelt sich um Geräte, die als CPUs den 8086, den 80286 und den 80386 verwenden. Leistungsmäßig an der Spitze steht das Modell M380 (mit 80386-CPU), das sich z. B. als Server in komplexen Netzwerken anbietet. Es läßt sich auf 52 MByte Hauptspeicher- und 270 MByte Festplatten-Kapazität ausbauen. Der Prozessor läuft mit einer Taktfrequenz von 20 MHz. An Betriebssystemen stehen neben MS-DOS auch Unix V/386, XENIX V/386 und MS-OS/2 zur Verfügung.

Als Server in einfacheren lokalen Netzen oder als intelligenter Arbeitsplatz mit mittlerer Leistung bietet sich das 80286-Modell M280 an. Ebenfalls mit dem 80286 ist das Modell S281 bestückt, das speziell als Netzarbeitsplatz entwickelt wurde.

Im unteren Bereich wird wohl der neue M240 den so erfolgreichen M24 ablösen. Wie sein Vorgänger arbeitet er mit dem Prozessor 8086 (10 MHz). Ein 20-MByte-Plattenlaufwerk sowie eine Diskettenstation mit 720 oder 360 KByte Kapazität sind eingebaut.



Der neue Computer M380/C von Olivetti

Datenbanksysteme

Auswahl und Einsatz: Wege zur individuellen Datenverwaltung. Von Alexander Janson. 186 Seiten mit 104 Abbildungen, Lwstr.-kart., 38 DM. Franzis-Verlag, München, 1987.
ISBN 3-7723-8671-7

Neben der Textverarbeitung ist die Arbeit mit Datenbanksystemen zu einem Schwerpunkt in der Nutzung von PCs geworden. Das Buch von Alexander Janson bietet eine ausführliche Einführung in die Theorie und Praxis der Datenbanksysteme. Nach einer auch für Einsteiger verständlichen Erklärung der wichtigsten Begriffe und Methoden werden die Komponenten der Hard- und Software mit zahlreichen Beispielen beschrieben. Der Leser lernt Grundoperationen wie Selektion und Projektion kennen und erfährt, wie man Datenbanken korrekt entwirft. Dabei hilft eine allgemeine Einführung in die Thematik der Dateiorganisation und eine Beschreibung weitverbreiteter Datenbanksprachen. Die Kaufentscheidung für ein bestimmtes Datenbanksystem wird duch Beurteilungskriterien erleichtert. Standard-Softwarepakete wie dBase II und dBase III, Symphony, Framework und Open Access werden beschrieben und untereinander verglichen.
Wer vor der Entscheidung steht, eine Datenbank aufzubauen, findet in diesem Buch die erforderlichen Kriterien zur richtigen Auswahl der Systemkomponenten. Neben Kapazitäts- und Geschwindigkeitsanforderungen werden im Kapitel Problemanalyse und Systemauswahl Hinweise gegeben, die vor falschen Entscheidungen bewahren. Praktische Beispiele in dBase II sowie Hinweise zur Datensicherung und zum Datenschutz runden das Buch ab. *Rs*

Geometrie

Methoden der grafischen und geometrischen Datenverarbeitung. Von Dr. Andreas Meier. 224 Seiten mit 93 Bildern und zahlreichen Beispielen zu Datenstrukturen und Algorithmen, kart., 38 DM, Teubner, Stuttgart, 1986.
ISBN 3-519-02482-9

Das Buch richtet sich an Interessierte verschiedener Ingenieurwissenschaften, welche eine Einführung in die grafische und geometrische Datenverarbeitung suchen. Der Bereich der Computergrafik umfaßt Geräte und Verfahren zur Beschreibung und Umwandlung von Daten in grafische Formen, während die geometrische Datenverarbeitung die Speicherung und Verarbeitung geometrischer Daten umfaßt. Das Buch basiert auf der entsprechenden Vorlesung der Abteilung für Informatik an der ETH Zürich und konzentriert sich auf die Datenstrukturen und Algorithmen, die in diesem Gebiet auftreten.
Es werden klassische Probleme der Computergrafik erläutert. Dazu gehören Transformationen, grafische Primitive und Operationen, Clipping sowie das Evaluieren verdeckter Kanten und Flächen. Selbst anspruchsvolle Themen wie z.B. mehrdimensionale Datenstrukturen zur Speicherung räumlicher Daten werden mit Beispielen, Zeichnungen und Listings (Modula-2) verständlich erklärt. Über eine Einführung in die Kurven- und Flächengeometrie spannt der Autor den Bogen zu komplexen dreidimensionalen Problemstellungen. Abschließend werden Entwicklungstendenzen der grafischen und geometrischen Datenverarbeitung aufgezeigt. In diesem Buch finden sich viele Lösungen zu grafisch/mathematischen Problemen, die in eigene Programme eingebaut werden können. *Rs*

Pascal

Algebra, Numerik, Computergrafik. Von Stephen Fedtke. Band 8 aus der Vieweg Programmothek, herausgegeben von Hansrobert Kohler. 324 Seiten, kart., 48 DM, Vieweg & Sohn Verlagsgesellschaft, Wiesbaden, 1987.
ISBN 3-528-04488-8

Die Liste der Themen beginnt bei der booleschen Algebra, und führt über die numerische Mathematik, die Codierungstheorie und Kryptographie bis hin zur grafischen Datenverarbeitung. Die Themen werden nicht nur praktisch anhand von Programmen abgehandelt, sondern deren mathematischer Hintergrund wird auch geschildert. So muß man sich zum Beispiel den Begriff boolesche Algebra nicht aus den in Definitionsfragen ziemlich soft formulierenden Technikerbüchern zusammenreimen, sondern bekommt ihn hier sauber mitgeliefert. Auch bei dem Studium der anderen Themen kann man sich gut an den Hinweisen zu den Grundlagen festhalten und so immer einen Weg in die Originalliteratur finden. Im Buch werden die Folgerungen für das Programmieren, die aus diesen Grundlagen resultieren, disktuiert und vor dem Hintergrund ausgearbeiteter Beispielprogramme beleuchtet. Die Beispielprogramme selbst haben Niveau, viele Grundaufgaben der praktischen Mathematik werden gelöst. Im Kapitel Codierungstheorie und Kryptographie ist zum Beispiel ein Programm enthalten, das die Verschlüsselung nach DES (Data Encryphtion Standard) nachvollzieht. Mit dieser Methode werden die gängigen Scheckkarten geschützt. Besonders interessant für Praktiker ist der Schluß des Buches. Dort kann man alles Grundlegende über Grafik mit Computern lernen und anhand von Pascalprogrammen ausprobieren. *Ro*

PC-Geheimnisse

IBM Insider-Tips. Von James E. Kelley. 245 Seiten, Diskette mit allen Hilfsprogrammen, kart., 69,80 DM, McGraw-Hill Book Company GmbH, Hamburg, 1986.
ISBN 3-89028-072-2

Der Autor hat sich die Mühe gemacht, die vielen kleinen Tips und Tricks (insgesamt sind es 127) rund um den IBM PC zu sammeln und thematisch zu ordnen. Als Leser werden diejenigen angesprochen, die mit Standard-Software auf PCs arbeiten und nun tiefer in die Materie eindringen möchten. Die vorgestellten Programme wurden durch das Übersetzerteam überarbeitet und an kompatible PCs angepasst. Auf der dem Buch beiliegenden Diskette sind alle vorgestellten Programme enthalten. Neben Basic-Programmen und DOS-Stapeldateien sind auch ablauffähige Assembler-Routinen enthalten, die teilweise das Arbeiten mit dem PC erleichtern. Den Hauptteil des Buches machen die Punkte PC-Hardware, Anpassung von Tastatur und Drucker und die Verbesserung der Bildschirmanzeige aus. Auch Tips zum optimalen Einsatz der Festplatte und Anregungen zum Erstellen von Stapeldateien gibt der Autor.
Anwender von Wordstar oder Lotus 1-2-3 erfahren, wie sie effektiver mit ihren Programmen arbeiten können. Allen Programmen, Tips und Tricks ist gemein, daß sie kurz sind und sehr ausführlich kommentiert werden. Das Buch wird durch zahlreiche Tabellen (Scancodes, BIOS-Index) abgerundet, die den fortgeschrittenen Anwender bei der weiteren Arbeit gut unterstützen.
Das Buch gibt viele praktische Tips, die dank der beiliegenden Diskette sofort in der Praxis angewandt werden können. *M. Donner*

Gerd Graf

Der mc-modular-AT

Teil 1: Der AT – wie Sie ihn haben wollen

Selbstbau, das scheint heute auf dem Feld der Mikrocomputer nicht mehr möglich zu sein. Wer kann schon Bauelemente sicher auf Multi-Layer-Platinen löten? Aber ohne Mehrschicht-Technik geht es heute nicht mehr, wenn man moderne Prozessoren verwenden will. Mit mc kommen Sie dennoch zu einem individuell Ihren Bedürfnissen entsprechenden Computer. Es ist ein modularer AT, dessen Aufbau und Eigenschaften in dieser Serie geschildert werden.

Im Januar 83 wurde der PC von IBM in Deutschland das erste Mal (18 Monate nach den USA) vorgestellt. Sein Erscheinen löste unterschiedliche Reaktionen aus, die von der totalen Ablehnung bis zur Euphorie reichten. Die technischen Daten des ersten Modelles, XT genannt, zeigt *Tabelle 1*. IBM hatte ein wichtiges Konzept bei der Einführung des PCs: die Offenheit im Hardware- und Softwarebereich. Bis heute ist dieses Konzept durchgehalten. Dies war sicher der Hauptgrund – neben der Marktmacht von IBM – für den heute bestehenden „IBM-Standard". Die Hardware und besonders die Bus-Schnittstellen waren von Anfang an offengelegt, was andere Hersteller dazu ermunterte, Baugruppen für den PC zu entwickeln und zu liefern. Dies war und ist für viele Hersteller interessant, da die ersten IBM-Geräte nicht mit allen benötigten Schnittstellen (z. B. Drucker, serielle Schnittstelle) ausgerüstet waren. Auf der Softwareseite entschied sich IBM für das Softwarehaus Microsoft, das damals im Mitbewerb zu Digital Research (CP/M 2.2) stand. MS-DOS, von IBM als PC-DOS angeboten, überzeugte durch eine klare Stuktur und einfache Bedienung. Durch die Offenheit des Systems regte IBM aber auch Zweitlieferanten an, IBM-kompatible Nachbauten herzustellen. Dies mag einerseits im Sinne des Herstellers gewesen sein, um den „IBM-Standard" weltweit durchzusetzen – heute hat sich jedoch dieser Markt zu einem gnadenlosen Preiskampfgebiet gewandelt. Das 1983 dargebotene Preis-Leisungsverhältnis hat sich dabei sicher um den Faktor 4-6 verbessert. Ein sehr einfacher IBM-PC-Nachbau ist heute schon zu Preisen um etwa 1000,– DM erhältlich.

Der AT ist aufwärtskompatibel

Mit der Vorstellung des IBM AT hat IBM eine weitere wichtige Marketingentscheidung getroffen: Die Investitionen an Software sind ungleich höher als die an neuer Hardware; es ist deswegen unbedingt nötig, zu bestehender Software aufwärtskompatibel zu sein. Der AT ist mit dem fortschrittlicheren 80286-Prozessor (siehe *Tabelle 2*) ausgerüstet. Der Prozessor wird jedoch nicht an die Grenze seiner Leistungsfähigkeit gefahren: er arbeitet im Real Address Mode voll kompatibel zum 8088. Damit ist ein Umstieg vom IBM PC auf den IBM AT praktisch problemlos möglich.

Die Kapazität des eingebauten Diskettenlaufwerks wurde auf 1,2 MB erhöht, die Platte vom AT 02 hatte 20 MB Kapazität. Die Diskettenlaufwerke sind jedoch problemlos in der Lage, 360 KB Disketten des IBM PCs zu lesen. Damit ist eine absolute Softwarekompatibilität gewährleistet. Disketten des IBM PC sind einfach in den AT einzustecken, das Programm lädt und es läuft wie bisher – nur wesentlich schneller. *Tabelle 2* zeigt die wichtigsten technischen Daten des AT.

Das Personal-System/2

Über das System /2 ist schon sehr viel geschrieben worden; interessant ist, daß IBM wiederum den Schutz der bestehenden Software-Investitionen ganz oben auf das Pflichtenheft gesetzt hatte. Beim

Tabelle 1: Die Leistungsdaten des IBM PC (1983)

```
- 256 KByte Hauptspeicher
- 40 KByte ROM
- Diskettenlaufwerk 360 KByte
- Raum für ein zweites Diskettenlaufwerk
- Diskettenadapter
- Prozessor 8088 von Intel
- 5 Erweiterungsplätze für Adapter
- Fassung für 8087
- Anschluß für Tastatur, Lautsprecher,
  Kassettengerät
- 250 ns Zugriffszeit
- 410 ns Zykluszeit
- Paritätsprüfung
- 4,77 MHz Taktfrequenz
- 130 W Netzteil

IBM XT FD Unterschied:
- Festplatte 10 MByte
- Festplattenadapter
- 8 Erweiterungsplätze für Adapter, drei belegt
```

Tabelle 2: Die Leistungsmerkmale des IBM AT 01/02

```
- Lesespeicher 64 KByte (ROM)
- Diskettenlaufwerk 1,2 MByte
- Festplatten-/Diskettenadapter
- 8 Erweiterungsplätze für Adapter
- Fassung für 80287 (Math. Co-Prozessor)
- Akkugepufferte Uhr/Kalender
- Prozessor 80286
- Real Address Mode (8086-kompatibel)
- Protected Virtual Address Mode
- Anschluß für Tastatur
- 7 DMA-Kanäle
- 16 Interrupt-Ebenen
- 3 programmierbare Timer
- 6,0 MHz Taktfrequenz
- 150 ns Zugriffszeit
- 355 ns Zykluszeit
- Netzteil 192 Watt
```

Tabelle 3: Die verschiedenen /2-Modelle

Modell	30	50	60	80	
CPU	8086	80286	80286	80386	
RAM	640	1000	1000	1000/2000	KByte
Diskette	720	1440	1440	1440	KByte
Platte I	20	20	20	44/70/115	MByte
Platte II	-	20	44/70/115	44/70/115	MByte

Design des PS/2 wurde das technologisch vernünftig Machbare angestrebt und nicht der Fortschritt um jeden Preis. *Tabelle 3* zeigt die wesentlichen Leistungsmerkmale der Modell-Familie /2. Die wichtigsten Unterschiede zum PC und AT liegen in der Technologie der Systemplatine, der integrierten Adapterfunktionen, des Micro-Channel-Designs, der integrierten Videographik und des 3½-Zoll-Datenträgers. Beim PS/2 setzt IBM Gate Array Chips ein, die sehr viele Komponenten ersetzen können. Dadurch werden der Integrationsgrad und die Ausfallwahrscheinlichkeit geringer (jeder Steckkontakt ist gefährlich!). Außerdem werden viele SMD-Bausteine verwendet. Vom Micro Channel bemerkt der „normale" Anwender nur den höheren Durchsatz, das schnellere Ablaufen seiner Programme. Die Videografik mit sehr hoher Auflösung erlebt er hautnah am Schirm; vor allem die Anwender des „einfachen" PCs mit der doch recht einfachen Grafik werden sich darüber freuen. Interessant ist, daß IBM 3½-Zoll-Disketten einsetzt, die sich damit sicher gegenüber der 3-Zoll-Diskette durchgesetzt hat. Sie besitzt höhere Kapazität gegenüber der 5¼-Zoll-Diskette (720 KB/1,2 MB), verbunden mit höherer Transport- und Berührungssicherheit. Wir gehen deshalb auf dieses Diskettenformat ein, da wir beim mc-modular-AT ebenso die Möglichkeit geschaffen haben, PS/2-Disketten direkt zu lesen sowie zu beschreiben. *Tabelle 3* zeigt die wichtigsten technischen Daten des PS/2.

Die Software unter PS/2

Parallel zum PS/2 haben IBM und Microsoft ein neues Betriebssystem für das IBM PS/2 angekündigt; es soll Ende des Jahres lieferbar sein. Wichtig ist, daß dieses Betriebssystem auch auf Kompatiblen mit 80286, also allen ATs und damit natürlich auch auf dem mc-modular-AT laufen soll. Das Betriebsystem/2 von IBM (bei Microsoft OS/2) ist zunächst zum bisherigen MS-DOS/PC-DOS völlig aufwärtskompatibel. Es ist ein Multitasking-Betriebssystem. PS/2 wird auch die bisherige Beschränkung im Festplattenbereich (max. 32 MByte unter MS-DOS) und im Arbeitsspeicherbereich (max. 640 KByte unter MS-DOS) aufheben. *Tabelle 4* zeigt die wichtigsten Leisungsmerkmale des IBM Betriebssystemes/2.

Die Anforderungen an den mc-modular-AT

PC-, AT- oder PS/2-kompatibel? – das war bei den Grundsatzgesprächen die wichtigste Frage. Für den PC-kompatiblen – also den mit der 8/16-Bit-CPU 8088 – sprachen vor allem Preisgründe. PC-kompatible werden heute sehr günstig angeboten. Man muß sich aber klarmachen, daß der Original-PC seit 1981 auf den Markt ist. Man kann also auch bei viel Wohlwollen nicht mehr von neuester Technologie sprechen! Die günstige Preisentwicklung vor allem bei Disketten- und Festplatten-Laufwerken hat dazu beigetragen, daß der Abstand vom PC zum AT-Kompatiblen im Preis heute nicht mehr allzu groß ist. Gegen den PC spricht besonders die geringe Kapazität bei den Diskettenlaufwerken. 360 KByte sind für normales Arbeiten und gelegentliche Backups bald zu wenig. Kopieren mit dem COPY-Befehl z. B. von Platte auf Diskette ist wesentlich einfacher und schneller durchzuführen als die doch recht umständliche Backup-Prozedur des Betriebssystems, die Dateien größer als 360 KByte auf mehrere Disketten verteilen kann. Es ist auch ein Unterschied, ob man mangels einer anderen Backup-Möglichkeit (Streamer oder beschreibbare optische Platte) zum kompletten Backup einer 20 MByte Platte etwa 20 (beim AT) oder etwa 60 Disketten (beim PC) benötigt! Ein weiterer Vorteil des AT gegenüber dem PC ist seine doch erheblich höhere Arbeitsgeschwindigkeit. Wer schon ein-

Tabelle 4: OS-2 wird auf dem AT laufen

Leistungsmerkmale des IBM Betriebssystem/2

- Zwei Modi, einen geschützten Modus und einen Kompatibiliätsmodus. Der Anwender kann per Taste zwischen beiden Modi hin- und herschalten oder per Menü ein anderes Programm aufrufen. Anschließend führt die Ausgabe-Steuerfunktion des Betriebssystem/2 die Modusänderung aus.
- Der geschützte Modus schöpft das Leistungsvermögen des Intel 80286 voll aus und eignet sich hervorragend für größere Anwendungen.
- Der Kompatibilitätsmodus arbeitet mit DOS ähnlichen Befehlen und unterstützt im Zusammenspiel mit TopView den konkurrierenden Einsatz von zwei oder mehr Anwendungs- oder Entwicklungsprogrammen. In diesem Modus laufen gut angepaßte, zeitunabhängige DOS-Anwendungen oder Programm-Entwicklungstools praktisch wie in einer DOS-Umgebung.
- Anwendungen im geschützten Modus können konkurrierend zu DOS-Anwendungen im Kompatibilitätsmodus ablaufen. Solange Anwendungen des geschützten Modus im Vordergrundbetrieb gefahren werden, wird der Kompatibilitätsmodus ausgesetzt.
- Volle Adressierbarkeit des Prozessors 80286
- Dynamische Speicherverwaltung
- Physische Hardfiles mit mehr als 32 MB werden in mehrere logische Laufwerke mit einer Maximalgröße von jeweils 32 MB untergliedert
- Prioritätsorientiertes Time-Sharingsteuerprogramm (Scheduler)

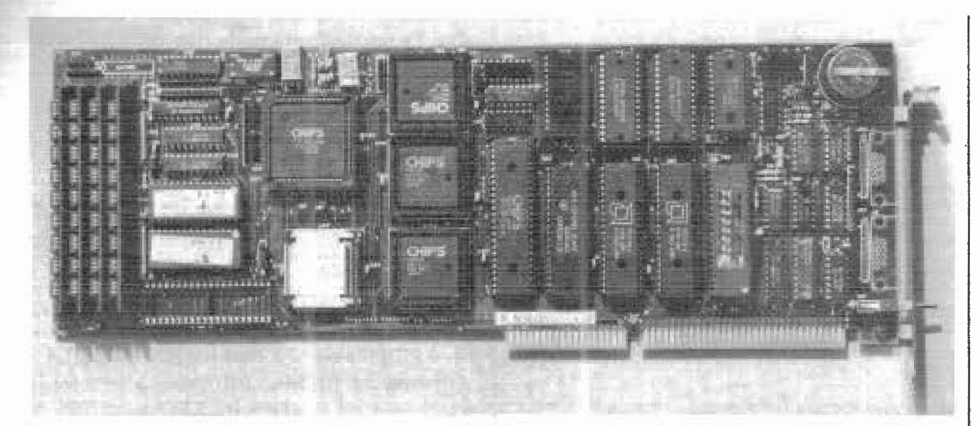


Bild 1. Der mc-modular-AT mit modernsten Chips

mal mit beiden Systemen gearbeitet hat, wird die Geschwindigkeit des AT nicht mehr missen wollen. Dies gilt besonders für rechen- und speicherintensive Arbeiten, wie z. B. der Compilierungs- und Linkvorgang beim Übersetzen. Die wichtigsten Argumente gegen den PC liegen aber in der Vorstellung der neuen IBM Serie (Personal System /2). Wichtig beim PS/2 ist der Umstieg von IBM auf das neue Datenträgerformat 3½ Zoll. Diese Disketten haben eine Speicherkapazität von 720 KByte netto (Modell 30) oder 1,4 MByte netto (Modelle 50-80). Das ausgewählte Rechnersystem soll zumindest in der Zukunft in der Lage sein, diese Diskettenformate zu unterstützen. Das Betriebssystem /2 soll ab Herbst an OEM-Kunden geliefert werden und bietet gegenüber dem „normalen" MS-DOS (bzw. PC-DOS) sehr viele Vorteile. Besonders wichtig ist die Multitasking-Fähigkeit, bei der verschiedene Aufgaben (tasks) quasi gleichzeitig auf dem Rechner laufen können. Dieses Betriebssystem läuft nur auf Rechnern mit der CPU 80286, also nur auf AT-Kompatiblen. Natürlich muß man auch in Betracht ziehen, daß IBM mit der Vorstellung des neuen PS/2 die „alten" PCs bis auf den AT 03 und den XT 286 nicht mehr herstellen wird. Deshalb haben wir einen modularen AT entwickelt.

Selbstbau kaum noch ökonomisch

Ein weiterer Punkt in der Diskussion war der Selbstbau. Gegen den Selbstbau sprachen ganz massive Gründe: Ein AT-kompatibler Rechner ist kein „einfaches" Objekt. Der komplette Selbstbau wäre nur für erfahrene Profis möglich gewesen. Die Anzahl der Bauelemente ist zwar durch die Verwendung von hochintegrierten Bausteinen aus der Produktion der Firma Chips Technologie stark reduzierbar – die Bausteine sind jedoch in bis zu 84poligen Gehäusen untergebracht. Diese Zahl an Anschlußbeinchen (im Abstand von 1,27 mm) kann nur auf einer mehrlagigen Leiterplatte (Multi-Layer) versorgt werden. Multi-Layer-Platinen sind wiederum mit dem Lötkolben nicht problemlos zu bearbeiten. Das schwerwiegendste Argument betrifft jedoch den Preis. Es werden heute eine Fülle von Kompatiblen-Baugruppen angeboten, die fast alle in Fernost hergestellt werden. Durch diese Marktschwemme, verbunden mit dem Verfall des Dollars, hat sich für all diese Baugruppen ein sehr günstiger Preis eingestellt. Wir haben es durchgerechnet: Die Zentraleinheit wäre, würde sie in Deutschland zusammengestellt, als Bausatz wesentlich teurer als das fertige, geprüfte Produkt aus Fernost. Ganz dem Selbstbau abgeschworen haben wir jedoch nicht: Die mc-multi-I/O-Karte, die Sie im Lauf dieser Serie kennen lernen werden, wird auch als leere Leiterplatte und als Bausatz zum Selbstbau erhältlich sein!

Das „modular" im Namen

Der mc-modular-AT heißt nicht nur so, weil „modular" gerade modern ist – er ist es tatsächlich. Die Zentraleinheit des mc-modular-ATs ist im Gegensatz zum Vorbild nicht auf einer großen Leiterplatte, dem Motherboard untergebracht, sondern auf einer Steckkarte, mit den Normmaßen einer langen AT-Baugruppe *Bild 1*. Dies wurde erst durch den Einsatz der hochintegrierten Bausteine von Chips Technologie möglich. Auf der Steckkarte befindet sich der gesamte Leistungsumfang des Motherboards – bei etwa 30 % der Fläche! Diese CPU-Baugruppe wird einfach in einen passiven IBM-Bus gesteckt. Auf dem passiven Bus befinden sich noch 7 weitere Plätze; diese sind wiederum völlig kompatibel zu den Buchsen auf dem Motherboard. Mit dieser Bauweise können recht kompakte Geräte aufgebaut werden. Auch der Einbau in ein 19-Zoll-Gehäuse bereitet keine Schwierigkeiten – ein wichtiges Merkmal für den industriellen Einsatz des mc-modular-ATs! Ein großer Vorteil dieser Modularität liegt im einfachen Austausch der CPU-Karte. Hier könnte später einmal eine Baugruppe mit dem 80386 – sofern diese vorhanden ist – eingesetzt werden. Durch den modularen Aufbau sind natürlich Fehlersuchen durch Kartentausch sehr schnell durchzuführen. Auch die Flexibilität spielt eine Rolle: da der Bus rein passiv ist, lassen sich für spezielle Anwendungen kürzere Busse herstellen. Eine Verlängerung ist allerdings aus elektrischen Gründen nicht ratsam.

Der Aufbau einen ATs

Grundsätzlich besteht ein IBM AT, wie fast alle seine Nachbauten, aus einem großflächigen Motherboard, in das man bis zu acht Erweiterungsplatinen (Ein/Ausgaben) einstecken kann. Auf dem Motherboard befindet sich die komplette Zentraleinheit mit den dazugehörigen Bauelementen. In die Erweiterungssteckplätze können Erweiterungsbaugruppen wie Festplatten- und Floppycontroller, Bildschirmcontroller, Druckeransteuerung ... eingesteckt werden. Bei den Einsteckplätzen wurde wieder die Aufwärtskompatibilität zum PC gewahrt. Acht Steckplätze entsprechen exakt der Belegung des IBM-8-Bit-Datenbus. In sechs dieser Steckplätze sind Karten einzustecken, die mit einem zweiten Buchsenplatz verbunden werden können. Über diese zweite Buchsenreihe werden die weiteren acht Datenbits des 80 286, sowie weitere Adreß- und Steuersignale des ATs zugeführt. Dadurch ist es möglich, in den IBM AT Steckbaugruppen des einfachen PCs aber auch die für den AT einzustecken. *Bild 2* zeigt den schematischen Aufbau der Basisbaugruppe eines AT. Der mc-modular-AT unterscheidet sich prinzipiell von diesem Aufbau: Die zentrale Einheit ist eine Steckbaugruppe. Die Busstecker, die sonst auf dem Motherboard untergebracht werden, sind auf einem rein passiven Bus untergebracht. Auf diesem Bus sind im wesentlichen nur die Datenleitungen der Steckplätze untereinander verbunden, sowie der Anschluß für das Netzgerät ausgeführt. Auf diese Weise ist ein wesentlich kompakterer Rechneraufbau möglich. Alle Signalleitungen sind aber vollkompatibel zum IBM-Motherboard. Deshalb können in den Bus des mc-modular-AT prinzi-

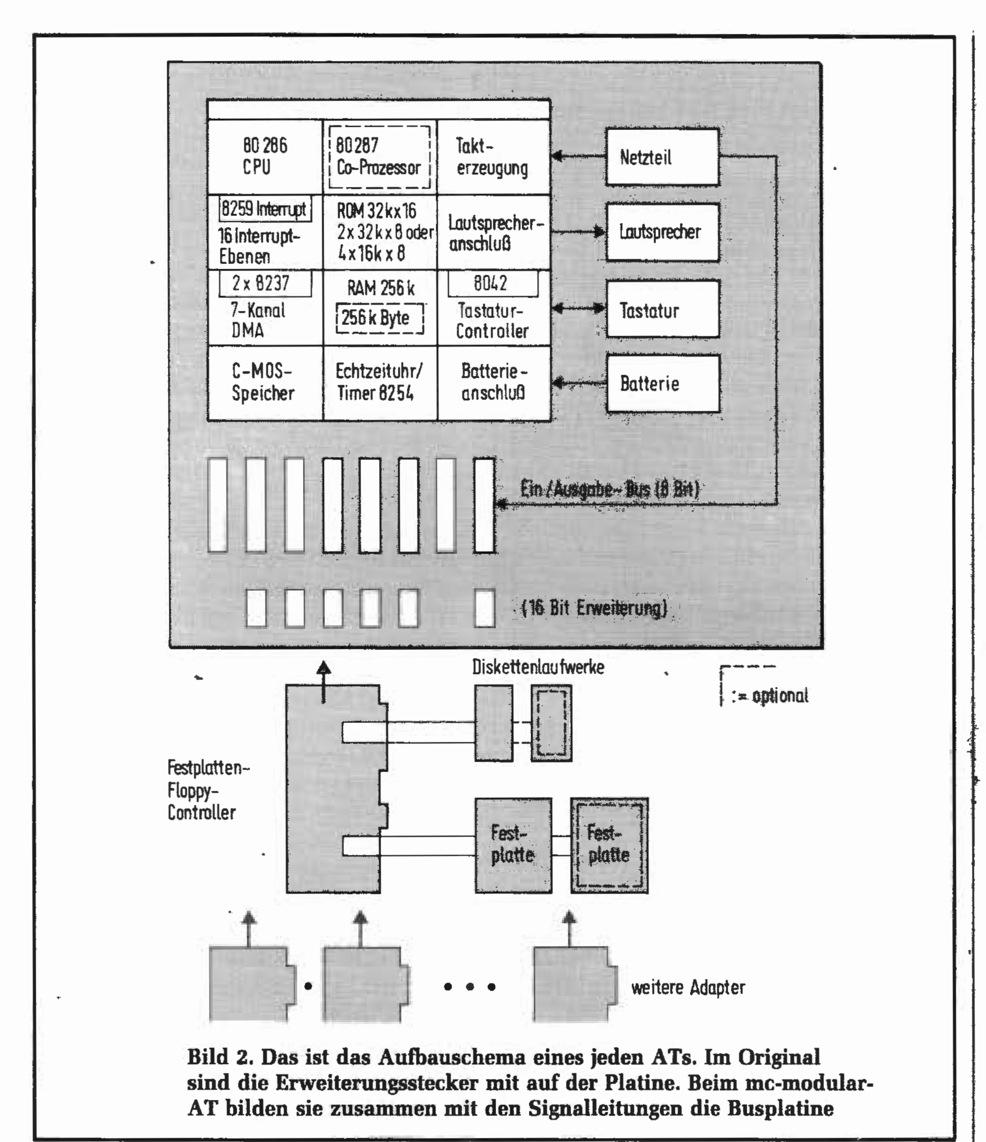


Bild 2. Das ist das Aufbauschema eines jeden ATs. Im Original sind die Erweiterungsstecker mit auf der Platine. Beim mc-modular-AT bilden sie zusammen mit den Signalleitungen die Busplatine

piell alle Baugruppen eingesteckt werden, die auch im normalen AT funktionieren.

Die Zentraleinheit

Das Blockschaltbild der Zentraleinheit ist im *Bild 2* dargestellt. Auf der Baugruppe befindet sich zunächst der Prozessor 80286, verbunden mit einem freien Einbauplatz für den Co-Prozessor 80287, der einfach dazugesteckt werden kann. Das ROM/PROM auf der Zentraleinheit besteht aus zwei 32-KByte- (oder vier 16-KByte-) ROMS oder PROMS und stellt 32 KWorte zu 16 Bit zur Verfügung. Darin befinden sich die Start- und Testroutinen sowie das BIOS des 286. Als RAM-Bausteine werden SIMM-Bausteine (Single Inline Memory Modules) verwendet. Auf der Zentraleinheit des mc-modular-AT ist bereits ein Speicherausbau von einem Megabyte vorgesehen. 512 KByte sind davon in der Grundversion bestückt. Das RAM ist 9 Bit breit, das neunte Bit dient zur Paritätsprüfung. Ein Paritätsfehler löst einen entsprechenden Interrupt aus. Als Timer ist ein Intel-Baustein 8254 eingesetzt. Die zwei Timer darin werden vorwiegend zur Erzeugung des Speicher-Refreshs sowie zur Ansteuerung des Lautsprechers verwendet. Zwei hintereinander geschaltete Interrupt-Controller (Intel 8259) stellen 16 Interruptebenen zur Verfügung, von welchen die Interrupts 3...7, 9...12, 14, 15 auf den Ein-Ausgabe-Bus geführt werden. Die restlichen Interrupts werden von der Systembaugruppe belegt.

Vom System werden sieben DMA-Kanäle unterstützt, die von zwei kaskadierten DMA-Controllern 8237 (Intel) erzeugt werden. Die DMA-Kanäle unterstützen einen schnellen Datentransfer von 8-Bit Ein-Ausgabe-Adaptern zum 8 oder 16 Bit breiten Systemspeicher. Jeder Kanal kann hierbei 64 KByte innerhalb des 16 MByte Systemspeicherbereiches transportieren. Als Echtzeituhr wird der Baustein MC 146818 von Motorola verwendet. Diese Uhr stellt eine Kombination aus Echtzeituhr und 64 Byte CMOS-RAM dar, das gepuffert werden kann. Bei einigen AT-Systemen ist noch weiteres CMOS-RAM auf der Zentraleinheit vorhanden. Im gepufferten CMOS-RAM werden die systemspezifischen Parameter hinterlegt, so daß diese nach der Initialisierung bei der Inbetriebnahme des Rechners normalerweise nicht mehr geändert werden müssen (unnormal: Batterie ist leer ...). Die Echtzeituhr sowie das CMOS-RAM werden über Akkumulatoren bzw. Lithiumbatterien mit Spannung versorgt. Mit auf der Systembaugruppe sind neben Takterzeugung, Lautsprecher und Batterieanschluß auch der Tastatur-Controller untergebracht. Beim IBM wird grundsätzlich eine serielle Tastatur eingesetzt. Die seriellen Daten werden meist von einem Single-Chip-Mikrocomputer (Intel 8042) decodiert und dem System zur Verfügung gestellt.

Nach Vorliegen der Tastaturdaten erzeugt der 8042 einen Interrupt (IRQ1, OUTPUT_BUFFER_FULL). Die Gesamtbelegung der Interrupts, sowie die detaillierte Belegung des Busses werden später exakt beschrieben.

Der Festplatten-Floppy-Controller

Der IBM AT ist nur mit dem Motherboard alleine noch nicht funktionsfähig. Zur Minimalkonfiguration werden neben dem Netzteil noch mindestens ein Festplatten/Floppy-Controller mit mindestens einem angeschlossenen Diskettenlaufwerk, sowie ein Bildschirmadapter benötigt. Beide Baugruppen werden in den Bus gesteckt. Der Festplatten-Floppyadapter hat, wie sein Name sagt, folgende Funktionen:

- Ansteuerung von 1...4 Floppylaufwerken
- Ansteuerungen von 1...2 Festplatten.

Ein AT muß mindestens mit einem Diskettenlaufwerk ausgebaut sein. Dieser Ausbau ist jedoch noch wenig sinnvoll. Bei den heute gesunkenen Preisen werden meist mindestens ein Diskettenlaufwerk sowie ein Festplattenlaufwerk mit 20 MByte Kapazität angeschlossen. Beim mc-modular-AT weist der Festplatten-Controller folgende Besonderheit aus: Es können auch Floppylaufwerke mit 3½-Zoll-IBM-PS/2-kompatiblen Disketten angesteuert werden. Interessant ist diese Besonderheit, da der

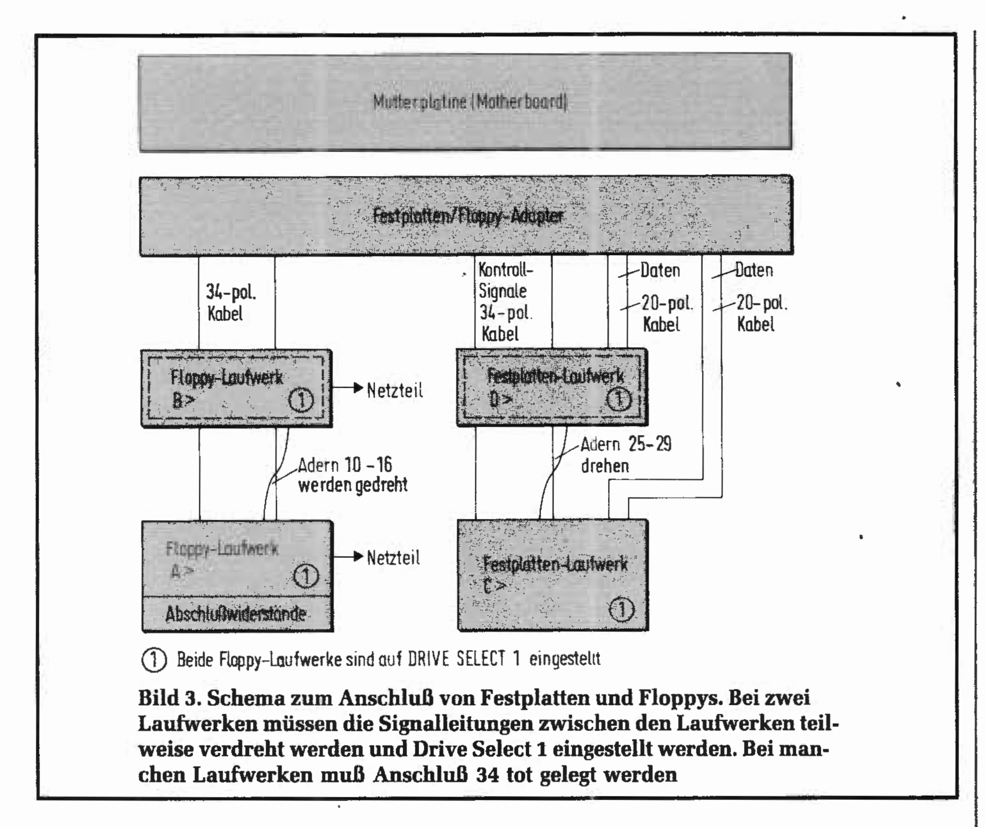


Bild 3. Schema zum Anschluß von Festplatten und Floppys. Bei zwei Laufwerken müssen die Signalleitungen zwischen den Laufwerken teilweise verdreht werden und Drive Select 1 eingestellt werden. Bei manchen Laufwerken muß Anschluß 34 tot gelegt werden

Adapter den gemischten Betrieb von 5¼-Zoll-Disketten und 3½-Zoll-Disketten vorsieht. Damit ist sowohl eine leichte Konvertierung von und nach beiden Formaten, sowie Kompatibilität zum bestehenden und zum zukünftigen Standard gewährleistet. Der mc-modular-AT ist ein moderner Computer! Am Festplatten-Floppy-Controller können eine oder zwei Festplatten angeschlossen werden. *Bild 3* zeigt schematisch die Standardanschlußbelegung von zwei Floppylaufwerken und zwei Festplatten beim IBM AT. Zu beachten ist, daß die Disketten und beide Festplattenlaufwerke auf Drive Select 1 eingestellt sind. Die unterschiedliche Decodierung erfolgt über das teilweise verdrehte Datenkabel! Beim Floppylaufwerk werden zwischen dem Laufwerk B und dem Laufwerk A die Adern 10-16, beim Festplattenlaufwerk zwischen D und zwischen C die Adern 25-29 gedreht. Sollen nur ein Diskettenlaufwerk und ein Festplattenlaufwerk angeschlossen werden, ergibt sich eine Anschlußbelegung gemäß *Bild 4*.

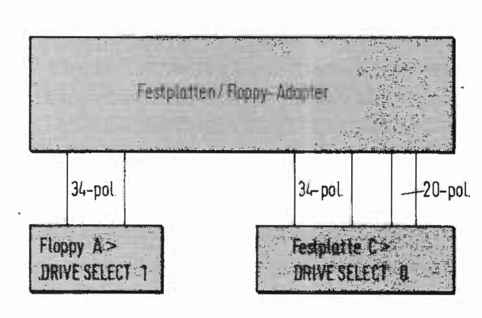


Bild 4. Wenn nur ein Laufwerk angeschlossen ist, dann muß jeweils Drive Select 0 eingestellt sein

Hier wird das Laufwerk einfach direkt angeschlossen. Es ist jedoch äußerst wichtig darauf zu achten, daß dann auf den beiden Laufwerken Drive Select 0 eingestellt wird. Leider halten sich in der Praxis nicht alle Floppy- und Festplattenlaufwerk-Hersteller an diese Belegung. Der Fehler äußert sich dann während des Bootvorgangs. Nach dem Speichertest – der am Bildschirm angezeigt wird – bleibt das System für ca. zwei Minuten „tot". In dieser Zeit wird mehrfach versucht, auf das Festplattenlaufwerk zuzugreifen. Ein mißglückter Zugriff wird durch den Fehler „Drive 0 not localised" angezeigt. Dieser Fehler deutet auf eine fehlerhafte oder falsch mit Drive Select versehene Festplatte hin. (Dies nur als Praxistip.)

Der Grafikadapter

Beim AT muß der Grafikadapter in die Systemeinheit (bzw. in den passiven Bus beim mc-modular-AT) eingesteckt werden. Der Vorteil dieses Verfahrens ist, daß man sich abhängig von seinen Wünschen und dem Geldbeutel einen passenden Adapter aussuchen kann. *Bild 5* zeigt die Übersicht über die gängigsten Grafikadapter, die im Lauf der Zeit entwickelt worden sind. Der einfachste Adapter ist der Monochromadapter (Monochrom steht für einfarbig). Er steuert einen Schwarzweiß-Monitor mit 80 Zeichen pro Zeile und 25 Zeilen mit dem IBM-Zeichensatz an. Durch die Möglichkeit, Blockgraphik nahtlos aneinander zu setzen, ist „Semi-Grafik" möglich. Dies bedeutet, daß zum Beispiel Tabellen nahtlos umrandet werden können. Der IBM-Zeichensatz enthält Elemente für eine Umrandung mit einem einfachen Strich, mit zwei Strichen und allen Kombinations- und Anschluß-

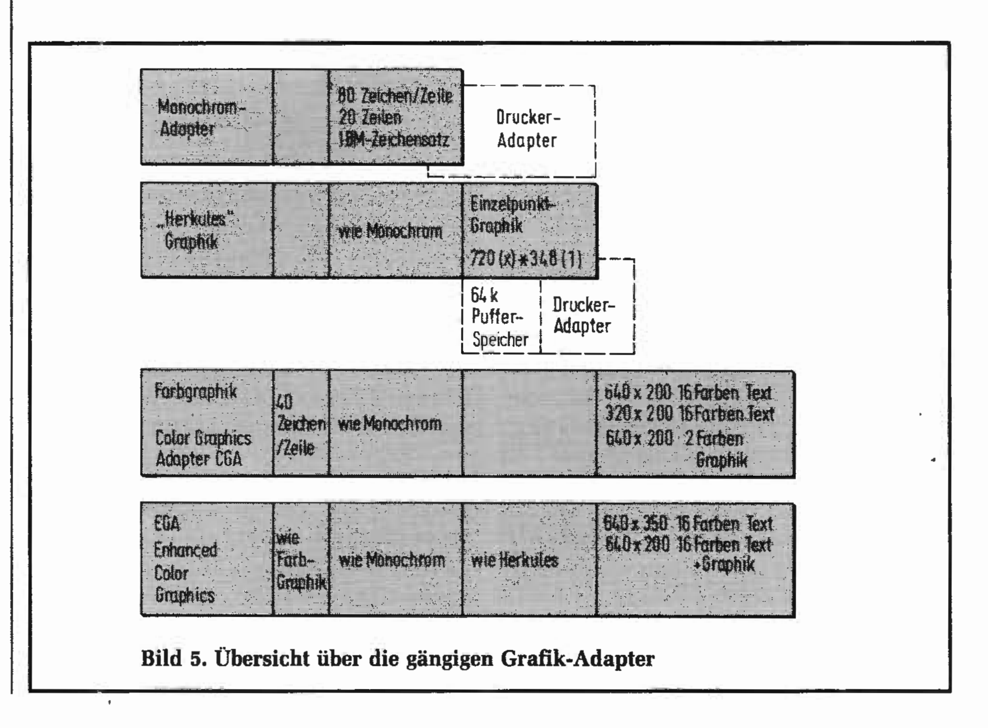


Bild 5. Übersicht über die gängigen Grafik-Adapter

elementen. Der Zeichensatz wird bei der folgenden detaillierten Beschreibung der Grafikkarte des mc-modular-ATs vorgestellt werden. Parallel zur Monochromgrafik wurde von IBM der Farbgrafik-Adapter (Color-Graphics-Adapter: CGA) vorgestellt. Diese Karte verhält sich einmal zunächst wie der Monochrom-Adapter, kann jedoch eine Darstellung in 16 Farben, entsprechend dem angeschlossenen TTL-Monitor zeigen.

Es können 640 × 200 Bildpunkte bei 16 Farben im Textmodus dargestellt werden. Bei der Grafikdarstellung mit 640 × 200 Punkten sind allerdings nur zwei Farben gleichzeitig möglich. Dieser Farbgrafikadapter wird heute kaum mehr eingesetzt. Einen Standard innerhalb des Standards zu setzen gelang der Firma Hercules mit der Vorstellung des Hercules-Adapters. Diese Herculesgrafik, wie sie auch kurz bezeichnet wird, ist kompatibel zur Monochromgrafik und kann parallel Einzelpunktgrafik von 720 × 348 Bildpunkten darstellen. Fast alle der heute gängigen Software-Produkte haben im Menü diese Herculesgrafik zur Auswahl. Deshalb haben wir die Herculesgrafik als preiswerte Grafik als Standard für den mc-modular-AT gewählt. Bei vielen der Hercules-Baugruppen ist noch ein Druckeradapter für den parallelen Drucker mit Centronics-Interface auf der Platine untergebracht. Weiter befindet sich meist auf der Baugruppe ein 64 KByte Pufferspreicher für die Grafikinformation. Die beste Kombination für den IBM – die allerdings zusammen mit dem benötigten Monitor etwas ins Geld geht – ist die EGA-Grafik. EGA steht für Enhanced Color Graphics. Die kann zunächst einmal alle vorhergenannten Grafikkarten emulieren. Es gibt heute EGA-Karten auf dem Markt, die die Umschaltung nicht über Jumper, sondern über Ports durchführen und es gibt einige EGA-Karten, die es zumindestens versuchen, diese Umschaltung automatisch zu machen. Neben der Emulierung der Hercules-Color- und Monochrom-Karte kann die EGA-Grafik auch eine relativ hohe Auflösung von 640 × 200 Punkten bei 16 Farben (gemischt Text und Grafik) darstellen. Die Auflösung bei nur Text beträgt 640 x 350, was im Vergleich zur Color-Grafik eine deutlich bessere Lesbarkeit des Textes ergibt.

Der Monochromadapter wird relativ selten eingesetzt: für nur Text- und einfache Grafikaufgaben verwendet man den Hercules-Adapter, für umfangreichere Grafik die EGA-Karte oder einer Ihrer Abkömmlinge.

Erweiterungen

Meist ist der Standard Centronics-Drukker bereits auf der Hercules-Karte untergebracht, so auch beim mc-modular-AT. Das System:

- Zentraleinheit
- Festplatten-/Floppyadapter
- Grafik/Druckeradapter

ist für einen Großteil der Anwendungsfälle sicher ausreichend. MS-DOS und das BIOS des IBM erlauben, daß bis zu drei Paralleldrucker angeschlossen (LPT1: bis LPT3:), sowie zwei serielle Schnittstellen (COM1:, COM2:) eingesteckt werden können. Mit einfachen Befehlssequenzen sind diese Schnittstellenkarten umschaltbar. Von IBM und weiteren Anbietern gibt es noch eine Reihe von weiteren sehr interessanten Karten. Sehr häufig wird ein sogenanntes „ABOVE-Board" verwendet. Hier werden 2 oder 4 MByte Speicher zusätzlich zur Verfügung gestellt. MS-DOS erlaubt es, einen fast beliebigen Teil des Speichers als RAM-Floppy zu deklarieren oder mit entsprechender Treibersoftware auf den größeren RAM-Speicher zuzugreifen. Weiter gibt es IEEE-Karten, AD/DA-Karten, Ein-/Ausgabe-Karten, EPROM- und PAL-Programmierer und natürlich Netzwerkadapter.

Im nächsten Teil der Serie wird es wesentlich technischer: Wir schildern die CPU-Baugruppe des mc-modular-ATs, sagen einiges über die dort verwendeten hochintegrierten Bausteine von Chips Technologie und beschäftigen uns detailliert mit dem IBM-Bus und seinen Signalen.

Resolution zur Computer-Arithmetik

Von den Grundoperationen +, –, ∗, / der Gleitpunkt-Arithmetik eines Computers verlangt man heute, daß das Resultat (außer bei Überlauf) für jede Wahl der Operanden „von höchster Genauigkeit" ist: Es muß übereinstimmen mit dem Ergebnis der Anwendung der arithmetik-inhärenten, eventuell vom Benutzer wählbaren, Rundung auf das exakte Resultat der Operation. Man vergleiche hierzu den IEEE Arithmetic Standard 754 (Mikroprozessor-Binärarithmetiken) und den Standard 854 (allgemeine Gleitpunkt-Arithmetiken).
Seit einiger Zeit vollzieht sich in hohem Maße eine Verlagerung des numerischen Rechnens vom Universalrechner auf Vektor- und Parallelrechner, sogenannte Supercomputer. Diese bieten neben den 4 Grundoperationen +, –, ∗, / in der Regel auch zusammengesetzte Operationen als weitere Grundbefehle an. Dies führt auf eine um Größenordnungen höhere Rechenleistung. Solche Grundbefehle gibt es beispielsweise für:

- multiply and add: a ∗ + c
- multiply and subtract: a ∗ b – c
- accumulate: berechnet die Summe der Komponenten eines Vektors
- multiply and accumulate: berechnet das innere oder skalare Produkt zweier Vektoren und andere.

Die GAMM fordert, daß alle Grundbefehle vom Hersteller so implementiert werden, daß garantierte Schranken für die mögliche Abweichung zwischen dem Gleitpunkt-Resultat und dem exakten Resultat mitgeliefert werden. Erstrebenswert und in der Regel erreichbar ist wiederum, daß das Endresultat einer solchen zusammengesetzten Gleitpunktoperation für alle Daten (die keinen Überlauf bewirken) mit dem gerundeten exakten Endresultat der zusammengesetzten Operation übereinstimmt. In diesem Fall kann auf die explizite Angabe einer Fehlerschranke verzichtet werden.

Der Benutzer sollte nicht gezwungen sein, für Grundbefehle, welche der Hersteller bereitstellt, bei jeder Anwendung eine Fehleranalyse durchzuführen.
Zur schnellen Berechnung zuverlässiger und möglichst enger Schranken in numerischen Algorithmen und zur Verifikation der Korrektheit berechneter Ergebnisse sollten alle Grundbefehle auch mit den gerichteten Rundungen zur Verfügung gestellt werden. Es muß in diesem Fall garantiert sein, daß das Gleitpunkt-Endresultat nur in der Richtung der gewählten Rundung vom exakten Endresultat abweichen kann. Für die skalaren arithmetischen Grundoperationen werden die gerichteten Rundungen bereits in den obengenannten Standards gefordert.

Gesellschaft für angewandte Mathematik und Mechanik (GAMM)

Lutz J. Koch

Universal-Interface

Teil 2: Parallel-Interface, 6522-, 6551- und 8255-Karte

Im ersten Teil wurde eine Adapter-Platine für den Apple II geschildert, welche die Signale verschiedener 8-Bit-CPUs so umformt, daß sie für die Interface-Platinen unseres universell verwendbaren Gerätes geeignet sind. Wer wirklich etwas mit seinem Computer steuern möchte, der benötigt mindestens eine der vorgeschlagenen Platinen. Unser Autor hat sie in einem Gerät zusammengefaßt, dem Universal-Interface.

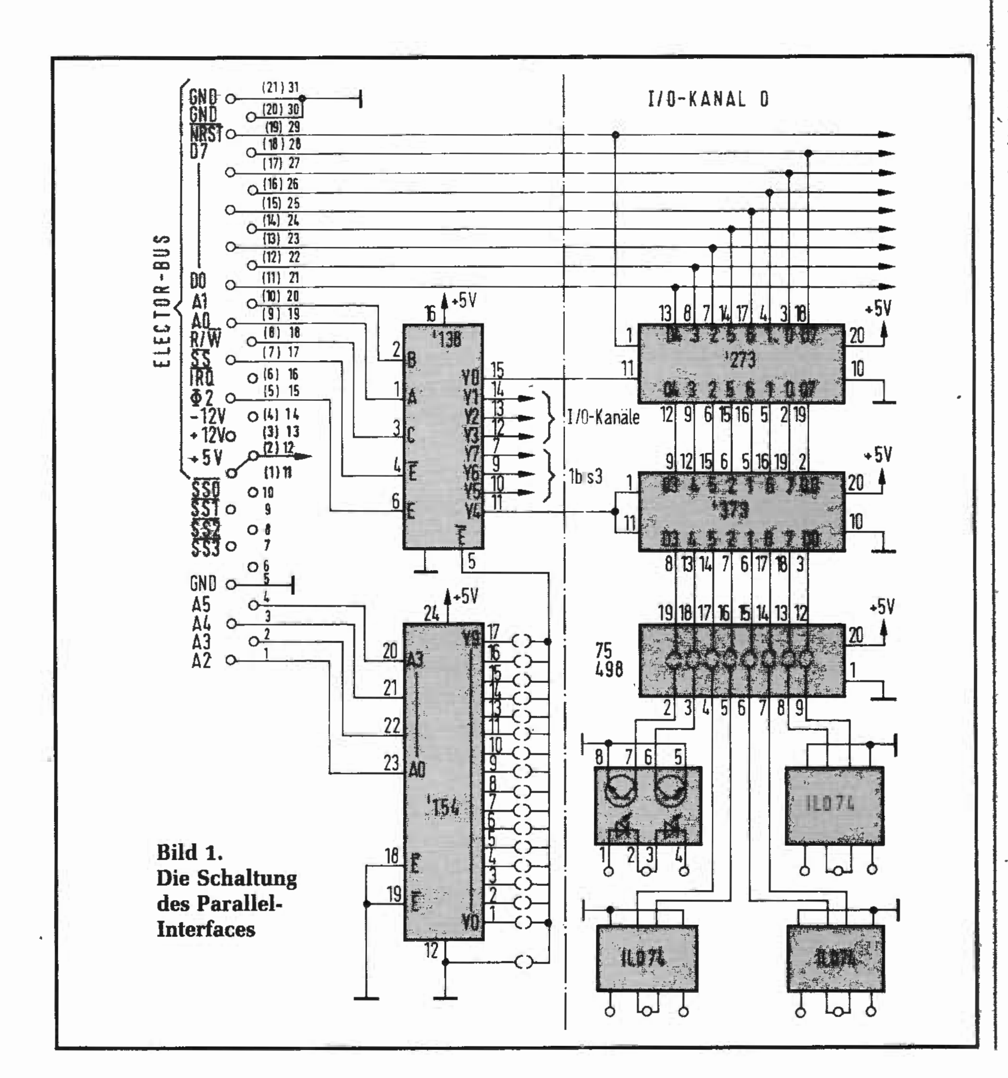


Bild 1. Die Schaltung des Parallel-Interfaces

Eines gleich vorweg: die Software-Bedienung der hier verwendeten Bausteine wird nicht geschildert. Wer darüber nicht informiert ist, der kann sie in den Handbüchern zu den Bausteinserien (8255 = Intel, 6522, 6551 = Rockwell) nachlesen. Das Parallel-Interface ist von der Logik her das einfachste. Also sei es zuerst geschildert.

Das Parallel-Interface

Auf der Platine (*Bild 1*) des Interfaces sind „nur" konventionelle Bausteine versammelt. Es besteht aus vier identischen Kanälen und der Decodier-Logik. Es setzt auf den Signalen des Elektor-Bussystems auf, das 256 I/O-Adressen bereit stellt, die durch die Signale $\overline{SS0}$...$\overline{SS3}$ in vier Bereiche zu je 64 Byte unterteilt sind. Der Baustein 74 138, ein Decoder mit acht Ausgängen, steuert genau eine der vier identischen I/O-Bausteingruppen an. Dabei verwendet er die Adressen A0 und A1 des Bussystems. Das Bussignal R/$\overline{W}$ wird ebenfalls über den '138er-Baustein geführt und entscheidet, ob gelesen oder geschrieben werden soll. Ein Baustein 74 154 (eins aus 16) decodiert die weiteren Adressen, die benötigt werden, um die Lage der Karte, die vier Byte beansprucht, im 64-Byte-Adreßraum festzulegen. Wenn man $\overline{SS}$ aus Bild 1 mit SS0 (siehe Teil 1, $\overline{SS0}$...$\overline{SS3}$ sind die Elektor-Slot-Select-Signale) verbindet, liegt das Parallel-Interface dann auf den Adressen XX...XX H.

Beachten Sie, daß der $\overline{SS}$-Anschluß (Nr. 17 bzw. 7) nicht mit denen anderer Karten verbunden werden darf!

Jede der vier Gruppen wird durch geeignete Bestückung entweder zum Eingang oder zum Ausgang. Steckt man in den Sockel, der dem Bus zunächst liegt, einen Baustein 74 273, der acht D-Flipflops mit Rücksetz-Eingang enthält, dann ist diese Gruppe auf Ausgang festgelegt und der Baustein 74 373 entfällt. An seinem Sockel müssen dann die Q-Pins mit den D-Pins verbunden werden, damit die im 273er zwischengespeicherten Bytes an das Treiber-IC 75 498 weitergegeben werden können, das acht Puffer mit offenem Kollektor-Ausgang (100 mA) beinhaltet. In dieser Situation dürfen die Optokoppler nicht eingesetzt werden; die Signale des Treiberbausteines sind dazu bestimmt, direkt in die Eingänge des zu steuernden Gerätes weitergeleitet zu werden. Dort sollten dann Optokoppler sitzen, falls diese benötigt werden.

Wenn die Gruppe als Eingang verwendet werden soll, darf der Baustein 74 273

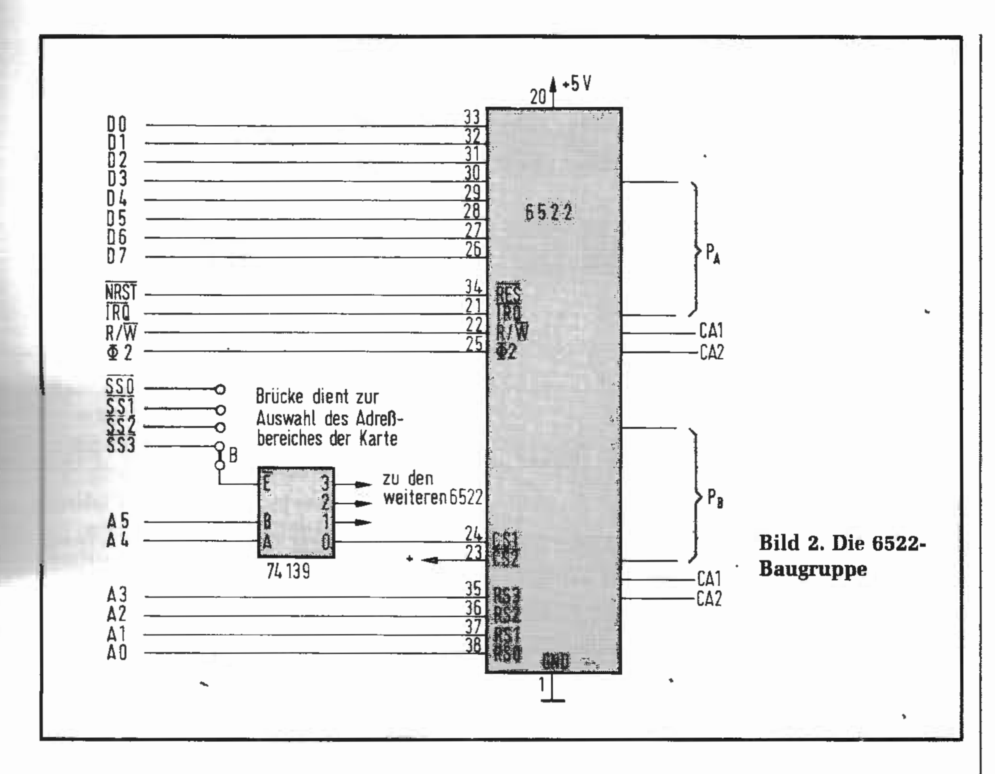


Bild 2. Die 6522-Baugruppe

nicht eingesteckt sein. Seine Q-Pins müssen mit den D-Pins verbunden werden, damit die Signale von der Peripherie an den Bus weitergeleitet werden können. Das IC 74 373 enthält acht flankengetriggerte D-Flipflops mit Tri-State-Ausgängen. Angesteuert wird es über vier Zweifach-Optokoppler, deren Ausgänge auf den Sockel für das IC 75 498 geführt sind. In diesen Sockel muß anstelle des ICs jetzt ein Widerstandsnetzwerk eingesetzt werden, das über 5 V die Eingänge des 373er bei gesperrten Foto-Dioden der Optokoppler auf eins hält. Anstelle des IC 75 498 sind dessen Ein- und Ausgänge mit Lötbrücken zu verbinden. Der Schaltungsvorschlag ist leicht abzuändern, falls Sie andere Bedürfnisse haben.

Die 6522-Baugruppe

6502-Fans ist der Baustein 6522 als VIA (Versatile Interface Adapter) bekannt. Dieser recht komplexe Baustein enthält zwei Ports mit je acht Bit, die zusätzlich mit Hand-Shake-Leitungen ausgerüstet sind. Außerdem zwei programmierbare Zeitgeber und Schieberegister zur Parallel-Seriell-Wandlung. Er kann sehr universell eingesetzt werden. Er besitzt 16 Steuer- und Datenregister, die alle nach bestimmten Vorschriften vorbelegt, beziehungsweise bedient werden müssen. Diese Vorschriften findet man im Handbuch zum Baustein. Es sei darauf hingewiesen, daß die Zeitgeber im 6522 den 6502-Systemtakt benötigen, der in Z80-Systemen nur schwer herzustellen ist (und mit der Adapter-Schaltung aus Teil 1 nur bei 6502-Systemen zur Verfügung steht). Die Platine, deren Schaltung *Bild 2* zeigt, enthält vier VIAs. Jedes dieser ICs belegt 16 Adressen, weshalb in Verbindung mit den Slot-Select-Signalen ein Vier-aus-Eins-Decoder 74 139 zur vollständigen Decodierung ausreicht. Sein Enable-Eingang wird mit einem der Signale $\overline{SS0}$...$\overline{SS3}$ verbunden, um die gewünschte Adreßlage herzustellen. Die Karte läßt sich leicht testen: Man schreibe in alle Register 0 und anschließend in eins der als Ausgang programmierten Datenregister einen beliebigen Wert. Wenn dieser Wert aus dem adressierten Register unverändert ausgelesen werden kann, ist mit großer Wahrscheinlichkeit alles in Ordnung.

Die 6551-Baugruppe

Der Baustein 6551 gehört ebenfalls zur 6502-Familie. Es ist ein asynchroner Sender/Empfänger für serielle Kommunikation mit eingebautem Baudraten-Generator. Mit dem richtigen Programm kann leicht eine V.24-Schnittstelle realisiert werden. In unserer Schaltung ist einer der vier Bausteine als Midi-Schnittstelle (*Bild 3*) eingesetzt. Die dafür notwendige Beschaltung mit Optokoppler und Treiber ist mit vorgesehen. Man beachte, daß wegen der Midi-Baudrate (31250 Bd) an dieser Stelle ein 3-MHz-Quarz verwendet werden muß. So ergibt sich mit einem programmierten Teilungsfaktor 96 der gewünschte Wert. Dies entspricht der Baudrate 19 200 Bd beim sonst zu verwendenden 1,8432 MHz-Quarz (*Bild 4*). Mit der 6551-Baugruppe besitzt man drei voneinander unabhängige V.24-Schnittstellen, denn es befinden sich jeweils zusätzlich noch die Schnittstellentreiber 75 188 und 75 189 auf der Platine. Man kann auch pinkompatible Vierfach-NANDs anstelle der Treiber einsetzen, falls man mit TTL-Signalen auskommt. Allerdings werden die 188/189-Bausteine über Pin 1 mit +12 V und Pin 14 mit −12 V verbunden. Bei anderen Bausteinen müssen Pin 1 abgekniffen werden und Pin 14 mit + 5 V verbunden wer-

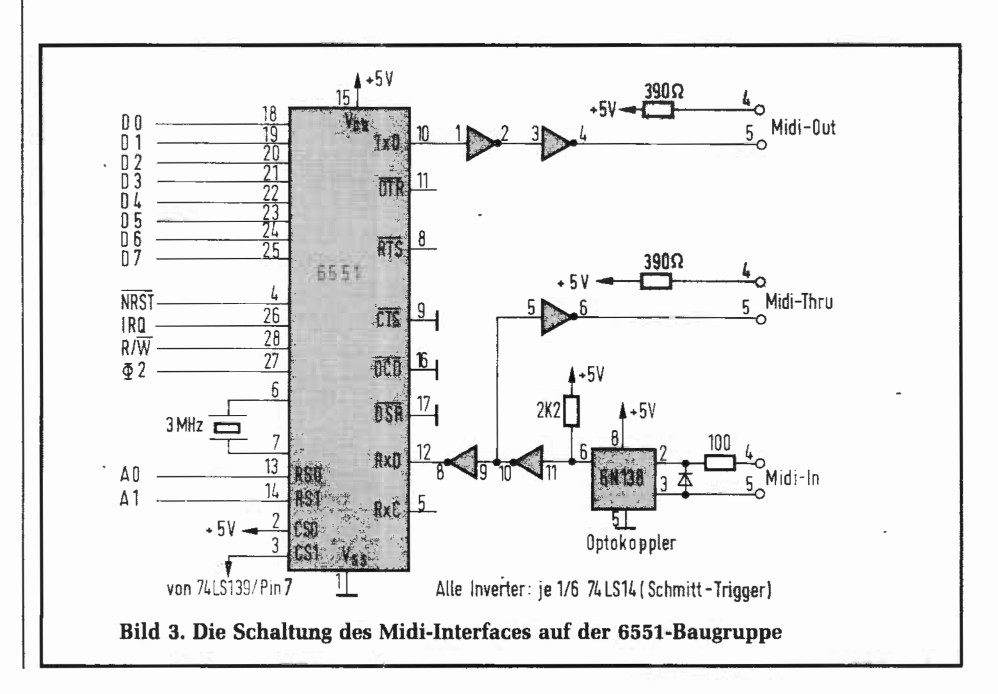
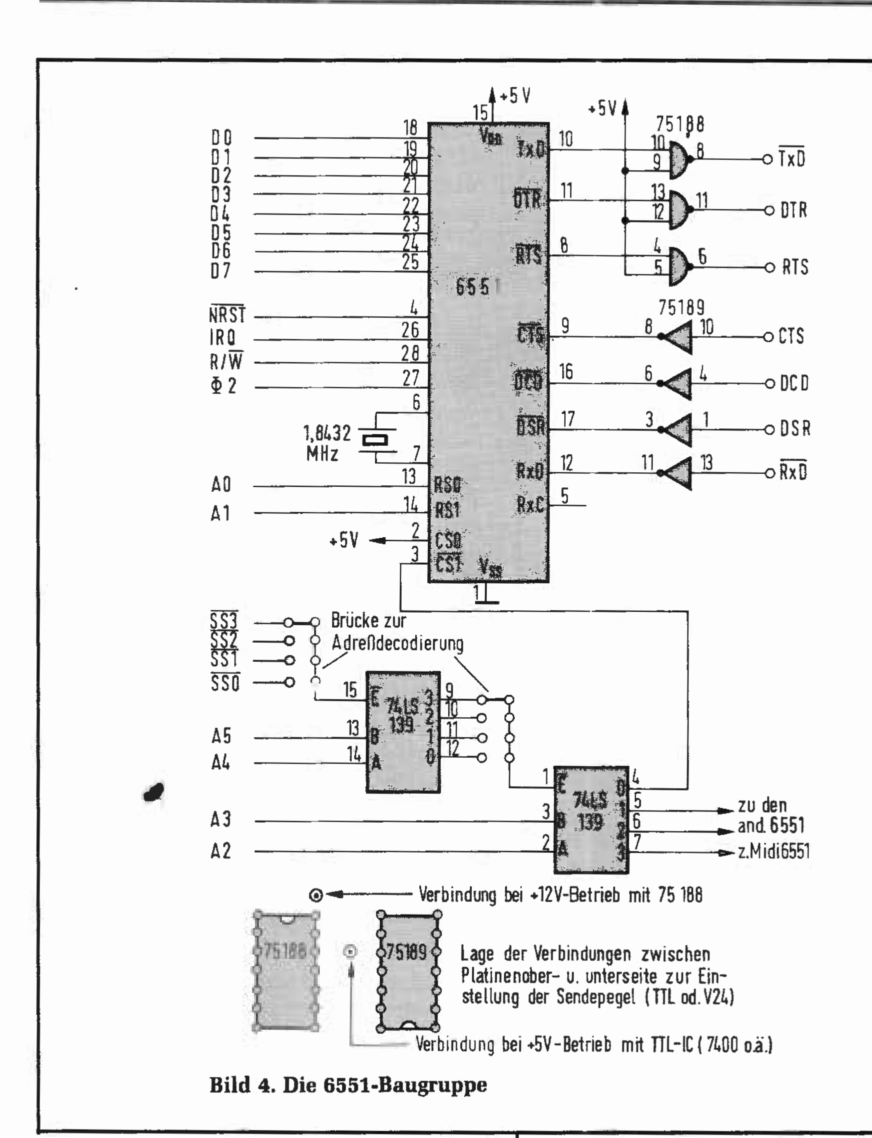
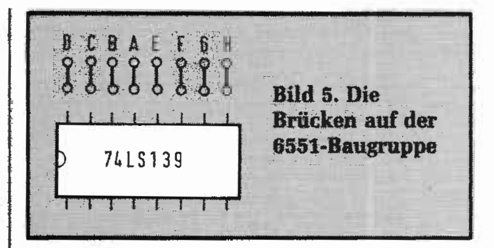


Bild 3. Die Schaltung des Midi-Interfaces auf der 6551-Baugruppe


Bild 4. Die 6551-Baugruppe

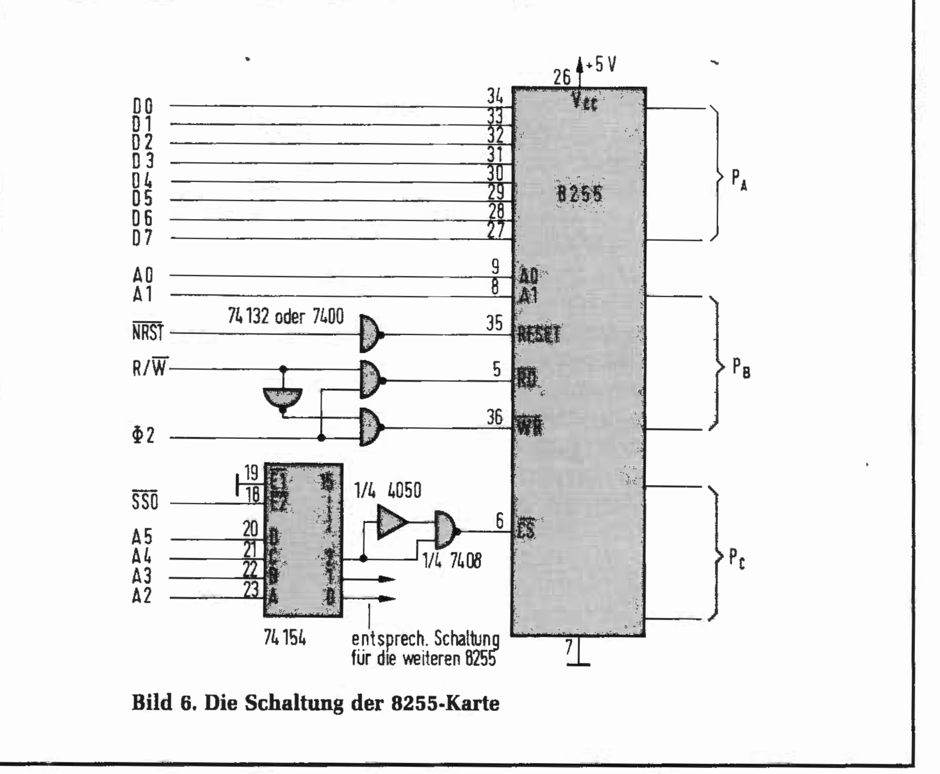

Bild 5. Die Brücken auf der 6551-Baugruppe

dem 6502 beruht, nicht direkt anschließbar. Es genügt aber, das Φ2-Signal mit R/W̄ so wie in *Bild 6* zu verknüpfen, daß die beiden Signale R̄D̄ und W̄R̄ entstehen. Danach ist nur noch das Reset-Signal zu invertieren und der Baustein wird richtig angesteuert. Auf der Platine sind drei 8255-Bausteine vorgesehen. Die Decodierung besorgt ein 74 154-Baustein. Da der 8255 vier Adressen belegt, können im Prinzip 16 Bausteine bedient werden. Die Karte ist so geschaltet, daß der Decoder wieder von einem der S̄S̄X̄-Signale angesteuert wird, hier von S̄S̄0̄. 04...0FH sind die Hex-Adressen der drei 8255. Für andere Adressen muß die Platine geändert werden. In der CS-Leitung zu den 8255-ICs ist jeweils eine Verzögerung eingebaut, bestehend aus einem CMOS- und einem 7408-Gatter, damit das Timing stimmt.
Die vorgestellten Schaltungen können auf Lochrasterplatinen in Fädeltechnik aufgebaut werden. Fertige Platinen liefern Neucom (München) und der Elektronikladen (Detmold).

den, wenn alles funktionieren soll. Die Platine ist für beide Fälle vorbereitet. Auf der Ober- und der Unterseite liegen sich entsprechende Lötaugen gegenüber, damit Sie nur mit kurzen Drahtstücken arbeiten müssen. Die Adreßdecodierung geschieht mit zwei 74 139-Bausteinen. Über eine Brücke kann man einstellen, auf welches Slot-Select-Signal die Karte ansprechen soll. Mit einer zweiten Brükke wird in 16er-Schritten eingestellt, welche Lage die 16 Adressen der Baugruppe innerhalb der 64 Adressen eines Slots einnehmen sollen. Jeder 6551-Baustein belegt 4 Adressen.
Die Brückenstellung zeigt *Bild 5*:
A = S̄S̄0̄, B = S̄S̄1̄, C = S̄S̄2̄, D = S̄S̄3̄, E = 0 ... 15, F = 16 ... 31, G = 32 ... 47, H = 48 ... 63. Die Bausteine 6551 müssen für 2 MHz ausgelegt sein.

Das 8255-Interface

Der Baustein 8255 stammt aus der 80er-Linie. Er ist an den Elektor-Bus, der auf


Bild 6. Die Schaltung der 8255-Karte

Jürgen Jordan

Turbo-Pascal bedient die EGA-Karte

Mit fünf Bytes sind Sie dabei ...

Haben Sie sich auch schon gewünscht, die in Turbo-Pascal eingebauten Grafikbefehle PLOT und DRAW sollten auch mit der EGA-Karte funktionieren? Wir stellen die notwendigen Änderungen vor und zeigen, wie man auch bei ähnlichen Problemen die richtige Stelle im Programm findet.

Alles begann damit, daß der Autor eines Tages einen IBM AT 02 mit EGA (64 KByte) und Monochrommonitor auf dem Dienstschreibtisch vorfand. Turbo-Pascal 3.0 lief auf Anhieb, aber leider ohne die eingebaute „Grundgrafik" (vgl. [1]), d. h. die Befehle GRAPHMODE oder HIRES [2] schalten die EGA-Karte (natürlich) nicht in den Grafik-Modus (0Fh: 640 × 350 Punkte monochrom) um. Wenn Sie mit DEBUG die Änderungen der *Tabelle 1* in TURBO.COM – oder

Tabelle 1: Fünf Bytes müssen geändert werden

Adresse	alter Wert	neuer Wert
Ø3F3	C7	5D
Ø3F4	ØØ	Ø1
Ø414	3F	7F
Ø415	Ø1	Ø2
Ø41C	Ø5	ØF

auch in TURBO-87.COM – vornehmen, leistet GRAPHMODE das Gewünschte, und auch die Anweisungen PLOT und DRAW funktionieren richtig, wenn auch nicht übermäßig schnell, wie eben alles, was den „Dienstweg" über die Interrupts nimmt.

Gewöhnlich sind Artikel dieser Art unter der Rubrik „Tips & Tricks" zu finden und an dieser Stelle schon zu Ende. Man tippt die paar Bytes ein und freut sich, daß es jetzt funktioniert. Dabei wüßte man aber eigentlich gerne, wie der Autor die zu ändernden Stellen gefunden hat, um sich bei einem ähnlichen Problem vielleicht selbst helfen zu können. Wie findet man also die Einsprungadresse

```
begin
  write('Anfang');
  graphmode;
  write('Ende');
end.
```

Bild 1: Das Testprogramm

der GRAPHMODE-Routine? Wer es lieber mag, kann stattdessen den HIRES-Befehl patchen.

Kein Problem, sollte man meinen. Wozu gibt es schließlich die Funktionen SEG und OFS, die nicht nur die Adressen von Variablen, sondern auch die von Funktionen und Prozeduren liefern? Leider Fehlanzeige, denn der Compiler meldet sich mit der Fehlermeldung „Unknown identifier ... " zurück. Mit etwas Nachdenken hätte man es vorher wissen können: SEG und OFS sind nur auf Objekte anwendbar, die im Deklarationsteil aufgeführt sind. Wir suchen jedoch die Adresse einer eingebauten Prozedur. Was nun? Die gesamte Turbo-Pascal-Laufzeitbibliothek (über 11 KByte) mit DEBUG „zu Fuß" zu durchsuchen, scheint weder verlockend noch aussichtsreich. Doch es geht viel einfacher: wir schreiben uns ein kleines Testprogramm (*Bild 1*), wobei wir die Anweisung GRAPHMODE zunächst weglassen, und compilieren mit COM-Option auf Diskette. Durch die beiden Strings 'Anfang' und 'Ende' finden wir den interessierenden Bereich mit DEBUG in TEST.COM schnell wieder (*Bild 2*). Natürlich dumpen wir erst ab 2D7CH, wie man dem Sprungbefehl an der Adresse 0100H entnehmen kann. Nun disassemblieren wir den durch 'Anfang' und 'Ende' markierten Bereich; am besten schalten wir dabei mit CTRL-P den Drucker zur Bildschirmausgabe parallel.

Jetzt fügen wir in das Testprogramm (*Bild 1*) den Befehl GRAPHMODE ein, compilieren und disassemblieren erneut. Sofort fällt auf, daß jetzt zwischen 'Anfang' und 'Ende' ein zusätzlicher Befehl steht, nämlich CALL 0418. Das muß der Aufruf der GRAPHMODE-Prozedur sein, und wir haben die Einsprungadresse gefunden.

Der Rest ist für Assemblerspezialisten einfach; wir Hochsprachenverwöhnten arbeiten uns mühsam durch das Unterholz und finden – hoffentlich – alle zu ändernden Stellen (*Bilder 3 und 4*). Durch den ersten Patch wird beim Aufruf von GRAPHMODE der Monochrom-Modus 0FH der EGA-Karte eingestellt. Ob der Farbmodus 10H funktioniert, konnte wegen des fehlenden Farbmonitors nicht ausprobiert werden. Die beiden anderen Patches passen die Maximalwerte für die Koordinaten x und y an. Das ist schon alles.

Mit den beschriebenen Änderungen läuft Turbo-Pascal beim Verfasser seit längerer Zeit einwandfrei. Und so sollte bei Ihnen jetzt auch das Programm

```
d 2D7C
1FØ3:2D7Ø                                         E8 57 DD Ø6               .W..
1FØ3:2D8Ø   ØØ 16 1B D6 24 DD Ø2 26-ØØ ØØ Ø4 ØØ A Ø 1Ø ØØ ØØ   ....$..&........
1FØ3:2D9Ø   ØØ ØØ ØØ 8B EC E8 34 DF-3Ø ØØ 9C 2D E9 ØØ ØØ E8   ......4.Ø..-....
1FØ3:2DAØ   B6 F7 E8 B8 FB Ø6 41 6E-66 61 6E 67 E8 73 E2 E8   ......Anfang.s..
1FØ3:2DBØ   66 D6 E8 A3 F7 E8 A5 FB-Ø4 45 6E 64 65 E8 62 E2   f.........Ende.b.
1FØ3:2DCØ   E9 ØØ ØØ 33 CØ E8 C1 DE-ØØ ØØ E8 DB FØ 29 E8 AØ   ...3.........)..
1FØ3:2DDØ   2Ø C6 Ø6 15 Ø6 ØØ 8Ø 3E-16 Ø6 ØØ 75 Ø7 8Ø 3E 1C    ......>...u..>.
1FØ3:2DEØ   Ø6 ØA 73 Ø4 E8 CF FØ 1E-E8 B9 1F AØ 1C Ø6 5Ø E8   ..s...........P.
1FØ3:2DFØ   FB 22 E8 77 ØD E8 A7 1D-58 5Ø 3A C1               .".w.....XP:.
```

Bild 2: So sieht der zu dem Programm aus Bild 1 gehörige Dump aus

```
1EEC:03E0 A30C00         MOV    [000C],AX
1EEC:03E3 C7060A000000   MOV    WORD PTR [000A],0000
1EEC:03E9 C7060E000000   MOV    WORD PTR [000E],0000
1EEC:03EF C70610000C700  MOV    WORD PTR [0010],00C7    ; 00C7 ersetzen durch 015D
1EEC:03F5 A00900         MOV    AL,[0009]
1EEC:03F8 32E4           XOR    AH,AH
1EEC:03FA CD10           INT    10
1EEC:03FC 33DB           XOR    BX,BX
1EEC:03FE 881E2000       MOV    [0020],BL
1EEC:0402 B40B           MOV    AH,0B
1EEC:0404 CD10           INT    10
1EEC:0406 FEC7           INC    BH
1EEC:0408 B40B           MOV    AH,0B
1EEC:040A CD10           INT    10
1EEC:040C 5D             POP    BP
1EEC:040D C3             RET
1EEC:040E C606090004     MOV    BYTE PTR [0009],04
1EEC:0413 B83F01         MOV    AX,013F                 ; 013F ersetzen durch 027F
1EEC:0416 EBC7           JMP    03DF
1EEC:0418 C606090005     MOV    BYTE PTR [0009],05      ; 05 ersetzen durch 0F
1EEC:041D EBF4           JMP    0413
1EEC:041F C606090006     MOV    BYTE PTR [0009],06
```

Bild 3: Der Bereich für die Ausgabe auf den Bildschirm aus TURBO.COM im Original

```
1F03:03E0 A30C00         MOV    [000C],AX
1F03:03E3 C7060A000000   MOV    WORD PTR [000A],0000
1F03:03E9 C7060E000000   MOV    WORD PTR [000E],0000
1F03:03EF C70610005D01   MOV    WORD PTR [0010],015D    <--- 3. Patch
1F03:03F5 A00900         MOV    AL,[0009]
1F03:03F8 32E4           XOR    AH,AH
1F03:03FA CD10           INT    10
1F03:03FC 33DB           XOR    BX,BX
1F03:03FE 881E2000       MOV    [0020],BL
1F03:0402 B40B           MOV    AH,0B
1F03:0404 CD10           INT    10
1F03:0406 FEC7           INC    BH
1F03:0408 B40B           MOV    AH,0B
1F03:040A CD10           INT    10
1F03:040C 5D             POP    BP
1F03:040D C3             RET
1F03:040E C606090004     MOV    BYTE PTR [0009],04
1F03:0413 B87F02         MOV    AX,027F                 <--- 2. Patch
1F03:0416 EBC7           JMP    03DF
1F03:0418 C60609000F     MOV    BYTE PTR [0009],0F      <--- 1. Patch
1F03:041D EBF4           JMP    0413
1F03:041F C606090006     MOV    BYTE PTR [0009],06
```

Bild 4: Derselbe Bereich nach dem Patchen

```
program scheibenwischer;

const   xmax = 640;     x_diff =-32;
        ymax = 350;     y_diff = 25;
        reverse = $8F;

var     x, y: integer;

begin
  graphmode;
  repeat
    x:=xmax + x_diff;      y:=0;
    repeat
      draw(0, 0, xmax, y, reverse);
      y:=y + y_diff;
    until y > ymax;
    repeat
      draw(0, 0, x, ymax, reverse);
      x:=x + x_diff;
    until x < 0;
  until keypressed;
  textmode;
end.
```

Bild 5: Der „Scheibenwischer" zum Ausprobieren

„Scheibenwischer" (*Bild* 5) laufen. Interessant an diesem Trivialprogramm ist eigentlich nur die verwendete „Farbe".

Mit „reverse" können Sie in eine bestehende Zeichnung hineinplotten, ohne Informationen zu verlieren: nochmaliges Plotten mit derselben „Farbe" – probieren Sie auch andere Werte aus – bringt die ursprüngliche Zeichnung wieder zum Vorschein. Beim nächsten Club-Treffen werden wir das gleich vorführen und auf diesbezügliche Fragen ganz lässig antworten: „Ist doch klar, die XOR-Verknüpfung ist assoziativ."

Literatur

[1] *Cebulla, U.:* Grafik mit der IBM-EGA, mc 1/87.

[2] Turbo-Pascal-Handbuch (Version 3.0), Heimsoeth-Software, München.

[3] *Byers, T. J.:* IBM PC AT, Mc Graw-Hill.

[4] *Michael, M.:* Enhanced Graphics Adapter, Markt & Technik.

Print Screen aus Basic

MS-DOS-Rechner besitzen eine PrtSc-Taste, mit der sich der Bildschirm-Inhalt jederzeit ausdrucken läßt – und zwar im Textmodus oder, sofern vorher das Hilfsprogramm GRAPHICS.COM resident geladen wurde, auch im Grafikmodus. Die zugehörige ROM-BIOS-Routine kann über INT 05H angesprungen werden. Möchte man aus einem laufenden Basic-Programm den Bildschirminhalt automatisch ausdrucken, so eignet sich hierfür ein nur drei Byte langes Maschinenprogramm mit den folgenden zwei Anweisungen:

INT 05
RETF

Im Basic-Interpreter ist es möglich, die resultierenden drei Objektcode-Bytes (hex CD 05 CB, dezimal 205 5 203) einer Stringvariablen zuzuweisen, die Anfangsadresse des Strings zu errechnen und dann mit CALL dieses Maschinen-Unterprogramm aufzurufen. Das *Bild* zeigt die so entstandene Print-Screen-Routine. Sie kann in beliebige eigene Programme eingebaut werden, wobei lediglich darauf zu achten ist, daß zwischen den Zeilen 110...140 keine weiteren Anweisungen eingebaut werden sollten, die die Adressenlage von Variablen im Datensegment verändern könnten. Die Anweisung in Zeile 130 ist hier theoretisch verzichtbar, wurde aber eingeführt, damit die CALL-Adresse im 2-Byte-Integer-Format vorliegt (±32767).

Herwig Feichtinger

Ausdruck des Bildschirminhalts in Basic

```
100 REM --- PrintScreen mit INT 05 ---
110 E$=CHR$(205)+CHR$(5)+CHR$(203):A=VARPTR(E$)
120 A=PEEK(A+1)+256*PEEK(A+2)
130 IF A>32767 THEN A=A-65536!
140 CALL A:REM Ruft INT 05 auf
```

Frank Ostrowski

Maschinen-routinen in GFA-Basic

Schnelleres Basic mit Assembler

In GFA-Basic ist es, wie in jeder Hochsprache, manchmal erforderlich, einen Teil des Programmes in Assembler zu schreiben. Die wichtigsten Gründe dafür sind der erhebliche Geschwindigkeitsvorteil und die größere Kontrolle über den Computer.

Es gibt mehrere Arten der Einbindung von Assembler-Routinen in ein GFA-Basic-Programm. Die einfachste Methode ist, die Bytes des Assembler-Programmes als DATA-Zeilen in das Basic-Programm aufzunehmen, und die Routine mit einer FOR-NEXT-oder WHILE-WEND-Schleife direkt in den Speicher zu POKEn. Ähnlich funktioniert das Verfahren, wenn das Programm nicht direkt in den Speicher, sondern in einen String oder ein Integerfeld geschrieben wird.

Dieses Verfahren eignet sich hervorragend für kleinere Routinen. Bei Routinen mit absoluten Adressen steigt der Aufwand aber schnell an, und auch der Zeitbedarf für das Einlesen der DATAs macht sich nachteilig bemerkbar. Außerdem belegen die DATAs Speicherplatz, obwohl sie nur ein einziges Mal zum Einlesen gebraucht werden.

Als nächstes Verfahren der Einbindung von Assembler-Routinen in Basic-Programme bietet sich das Einlesen der entsprechenden Daten von Diskette mit BLOAD oder über INPUT$ oder BGET an. Gegenüber der DATA-Methode verringert sich der Speicherplatzbedarf der Routinen, da sie auf einem Massenspeicher gehalten werden. Außerdem, und das ist sehr wichtig, verringert sich der Arbeitsaufwand bei Programmänderungen. Bei diesem Verfahren bleibt der Nachteil der schwierigen Behandlung absoluter Adressen allerdings bestehen und das Nachladen der Routinen dauert doch sehr lange.

Etwas komfortabler funktioniert die Einbindung der Assembler-Routinen mit dem GFA-Basic-Befehl EXEC, der folgende Syntax hat:

EXEC 0,"XXX.PRG","Kommandozeile","Environment"

Mit diesem Befehl wird eine Assembler-Routine oder ein kompiliertes Programm aufgerufen. Beim Laden werden automatisch die Adressen angepaßt. Auch dieser Aufruf ist, meine ich, nicht immer optimal. Zum Sortieren eines Stringfeldes würde in einem Programm etwa stehen:

EXEC 0,"SORT.PRG",STR$(*α$()),""

Ein offensichtlicher Nachteil des EXEC-Befehls ist die erzwungene Übergabe der Feld-Adresse als String. Die geladene Routine muß zumindest eine Umwandlung von ASCII-Text in Zahlen vornehmen können. Vorher muß außerdem noch mit RESERVE ausreichend Speicherplatz freigegeben worden sein. Ein weiterer Nachteil ist, daß das Programm bei jeder Benutzung von der Diskette oder der Festplatte nachgeladen wird. Arbeitet man mit diesem Verfahren ohne RAM-Disk, hat man einen Grund mehr, eine Kaffeepause einzulegen.

Erhebliche Probleme ergeben sich, wenn mehrere Dateien nachgeladen werden (z.B. bei Verwendung des GDOS). Die Speicherverwaltung des TOS scheint nicht immer den nicht benötigten Speicherplatz korrekt wieder freizugeben. Das äußert sich dann so, daß nach etwa 20 Ladevorgängen plötzlich nichts mehr geht – oft nicht einmal mehr das Öffnen einer Datei, z.B. zum Sichern des Programms auf Diskette.
Es gibt auch noch die Alternative, mit EXEC(3,...) ein Programm nachzuladen und ständig im Speicher zu halten. Dies erfordert aber eine Menge Aufwand bei der Programmerstellung.
In reinem Assembler ist das nicht weiter schwierig, aber für jeden C-Compiler sind andere Klimmzüge nötig. Das Standard-Startup-File (das File, das zuerst gelinkt wird), muß geändert werden. Hier gibt es außerdem, besonders in der Testphase, erhebliche Probleme mit der Speicherverwaltung des Betriebssystems.

Die Lösung: MONITOR

Zur Lösung der vorgenannten Probleme gibt es einen unscheinbaren Befehl im GFA-Basic: MONITOR.
Dieser Befehl übergibt einen optionalen Langwort-Integer-Parameter vom Register D0 an den Illegal-Instruktion-Handler (vier Bomben) durch Ausführung des Assembler-Befehls $4AFC (illegal, Breakpoint in Monitorprogrammen).
Nun kann der zugehörige Vektor auf eine selbstgeschriebene Routine umgebogen werden.
Das Beispielprogramm (*Bild 1*) verwendet diese Technik, um eine Sortierroutine für Stringfelder bereitzustellen. Der Ablauf dieses Programms ist wie folgt:

– Speicherfreigabe über

mshrink (gemdos(74));

wird in C durch das erste .O-File beim Linken erledigt.

– Verbiegen des Vektors (auch in C) über

supexec (xbios(38)).

Wichtig ist, daß der Wert des alten Vektors gerettet wird.

– Ausführung des Hauptprogramms (GFA-Basic-Interpreter oder kompiliertes GFA-Basic-Programm) über

pexec (gemdos(0x4b)),

beim Aufruf des Interpreters wird der Name des auszuführenden Programmes als Kommandozeile übergeben.

```
even    macro
        ds.w    0
        endm

term    equ     0       ;gemdos
conin   equ     1       ;gemdos
conwrs  equ     9       ;gemdos
mshrink equ     74      ;gemdos
pexec   equ     $4B     ;gemdos

supexec equ     38      ;xbios


        section text

beginn  bra.s   start
nam     dc.b    'GFABASIC.PRG',0
        ds.b    nam+20-*
cmd     dc.b    11              ;laenge
        dc.b    'MC_SORT.BAS',0
        ds.b    cmd+21-*
env     dc.w    0

* so laesst sich das Programm ohne neue Assemblierung anpassen:
*
* OPEN "U",#1,"MC_SORT.PRG"
* p$="GFABASRO.PRG"             !oder mc_demo.prg
* c$="MC_DEMO.BAS"              !wenn kompiliert
* SEEK #1,30
* PRINT #1,p$;CHR$(0);
* PRINT #1,CHR$(LEN(c$));c$;CHR$(0);
* CLOSE #1
*
* maximal je 20 Zeichen
*


start   lea     ende(pc),sp             ;Stackpointer umsetzen
        move.l  #ende-beginn+256,-(sp)  ;Speicherfreigabe
        pea     beginn-256(pc)
        clr.w   -(sp)
        move.w  #mshrink,-(sp)
        trap    #1
        adda.w  #12,sp

        pea     init
        move.w  #supexec,-(sp)
        trap    #14
        addq.l  #6,sp

        pea     env(pc)         ;exec 0,nam$,cmd$,env$
        pea     cmd(pc)
        pea     nam(pc)
        clr.w   -(sp)
        move.w  #pexec,-(sp)
```

Bild 1. Sortierprogramm für GFA-Basic in Assembler

```
        trap    #1
        adda.w  #14,sp
        move.w  d0,-(sp)

        pea     ex.init
        move.w  #supexec,-(sp)
        trap    #14
        addq.l  #6,sp

        move.w  (sp)+,d0
        beq.s   ok
        pea     fehl.txt(pc)    ;Fehlermeldung ausgeben
        move.w  #conwrs,-(sp)
        trap    #1
        addq.l  #6,sp

        move.w  #conin,-(sp)    ;auf Tastendruck warten
        trap    #1
        addq.l  #2,sp

ok      clr.w   -(sp)           ;Programm beenden
        trap    #1

*********************************
*                               *
*       Vektor verbiegen        *
*       (Supervisormode)        *
*                               *
*********************************

init    move.l  4*4,vec.ill
        move.l  #monitor,4*4
        rts

ex.init move.l  vec.ill(pc),4*4
        rts

*********************************
*                               *
*         MONITOR *tool%        *
*                               *
*********************************

monitor move.l  d0,a0                   ;Variablenadresse
        move.l  #dispatcher,(a0)        ;Routinenadresse
        addq.l  #2,2(sp)                ;Stackkorrektur fuer $4AFC
        rte                             ;zurueck

*********************************
*                               *
*       Sprungverteiler         *
*                               *
*********************************
*                               *
*       C:tool%(func,......)    *
*                               *
*********************************
```

```
dispatcher move.w 4(sp),d0        ;Funktionsnummer
        beq.s   sortier           ;bei mehr Routinen Sprungtabelle
error   moveq   #-1,d0            ;-1 als Fehlermeldung
        rts

*********************************
*                               *
*     Die Sortierroutine        *
*                               *
*        Shell-Sort             *
*                               *
*********************************
*                               *
*  VOID C:tool%(0,L:*a$(),off)  *
*                               *
*********************************

sortier move.l  6(sp),a0          ;*a$()
        move.l  (a0)+,a1          ;Anfangsadresse des Feldes
        cmp.w   #1,(a0)           ;eindimensional?
        bne.s   error             ;nein --> Fehler

        move.l  (a1)+,d0          ;Anzahl Feldelemente

        move.w  10(sp),d2         ;Vergleichsoffset
                                  ;shell-sort

        move.l  d0,d7

loop    lsr.l   #1,d7
        beq.s   fertig
        move.l  d7,d1             ;d1=d7*6
        add.l   d1,d1
        add.l   d7,d1
        add.l   d1,d1
        move.l  d7,d6
fori
        move.l  d6,d5
nextj   sub.l   d7,d5
        bmi.s   nexti

        move.l  d5,a5             ;a5=d5*6
        adda.l  a5,a5
        add.l   d5,a5
        adda.l  a5,a5
        add.l   a1,a5             ;+basis

        lea     (a5,d1.l),a6

        move.l  (a5)+,a3          ;adresse string 1
        move.w  (a5)+,d3          ;laenge string 1

        move.l  (a6)+,a4          ;adresse string 2
        move.w  (a6)+,d4

        sub.w   d2,d3             ;offset
        sub.w   d2,d4             ; Vergleichsbeginn bei Stringoffset
        adda.w  d2,a3             ; IF MID$(a$,off)>MID$(b$,off)
        adda.w  d2,a4
                                  ; Stringvergleich (a3),(a4)
vglstr  subq.w  #1,d4             ;string 2 zu Ende?
        bmi.s   vglx2             ;--> gleich oder string 1 laenger
        subq.w  #1,d3             ;string 1 zu Ende?
        bmi.s   vglx1             ;--> string 2 ist groesser!

        cmpm.b  (a3)+,(a4)+       ;vergleich
        beq.s   vglstr            ;noch gleich

        bhi.s   nexti             ;string 2 groesser, nicht vertauschen

swap    move.w  -(a5),d3          ;strings vertauschen
        move.w  -(a6),(a5)
        move.w  d3,(a6)
        move.l  -(a5),d3
        move.l  -(a6),(a5)
        move.l  d3,(a6)
        bra     nextj

vglx2   subq.w  #1,d3             ;gleich?
        bpl.s   swap
vglx1                             ;string 1 kuerzer
nexti   addq.l  #1,d6
        cmp.l   d0,d6
        blo.s   fori
        bra     loop

anpass  move.l  (a1)+,a0          ; Jetzt den Trailer anpassen,
        move.w  (a1)+,d1          ; damit das BASIC nicht
        beq.s   fertig            ; abstuerzt
        ext.l   d1
        addq.w  #1,d1             ;naechstgroessere gerade Zahl
        and.w   #$FFFE,d1
        lea     -6(a1),a2         ;Descriptoradresse
        move.l  a2,(a0,d1.l)      ;einsetzen
fertig  dbra    d0,anpass
        sub.l   #$00010000,d0     ;koennen auch mehr als 65535 Strings sein.
        bpl.s   anpass
        rts

*       Hier muesste ein .DATA stehen, dies kann der, sonst gute,
*       GST-Assembler aber nicht. Macros waeren zu lang.

fehl.txt dc.b   13,10,9,'Fehler beim Nachladen'
        dc.b    13,10,9,'Eine Taste druecken',0

        even

*       Hier muesste ein .BSS stehen, s.o.

vec.ill equ     *

ende    equ     vec.ill+4+256     ; Stackspeicher fuer Hauptprogramm

        end
```

```
'
' Demonstrationsprogramm für mc_sort
'
MONITOR *tool%                          !Toolkit Adresse bestimmen
sort%=0                                 !Funktionsnummer Sortierung
'
n=20                                    !Stringanzahl
DIM a$(n)
t1%=TIMER                               !Zeitnahme 1
FOR i%=0 TO n
  a$(i%)=STR$(RND)                      !Zufälliger Inhalt
NEXT i%
t2%=TIMER                               !Zeitnahme 2
VOID C:tool%(sort%,L:*a$(),4)           !Sortieren, skip 4 Zeichen
t3%=TIMER                               !Zeitnahme 3
FOR i%=0 TO n-1
  PRINT a$(i%)                          !Kontrollausgabe
NEXT i%
PRINT (t2%-t1%)/200;" s Feldaufbau"
PRINT (t3%-t2%)/200;" s Sortierung"
```

Bild 2. Das Sortierprogramm aus Bild 1 wird angewandt

– Nach der Rückkehr zum Assembler-Programm („QUIT" aus GFA-Basic) wird der Vektor restauriert, bei fehlerhaftem pexec wird eine Fehlermeldung ausgegeben.

– Programm wird durch

(term, gemdos(0))

beendet.

Wenn das GFA-Basic einen MONITOR-Befehl durchführt, wird an der Adresse, auf die der Wert in Register D0 zeigt, die Startadresse des Unterprogrammverteilers abgelegt. Beim Testen dieser Routine ist Vorsicht geboten. Der Aufruf muß immer mit

*MONITOR *tool%*

erfolgen, der Zeiger auf die Integervariablen steht also in D0.

– Der Exeption-Handler besteht aus vier Assembler-Befehlen:

move.l d0,a0
move.l #dispatcher,(a0)
addq.l #2,2(sp)
rte

Mit dem ersten Befehl wird das Register D0 als Zeiger belegt. Der zweite Befehl dient zum Einschreiben der Routinen-Adresse in der Integervariablen, worauf die Rücksprungadresse auf dem Stack um zwei erhöht wird. Beim Rücksprung über rte wird dadurch *illegal ($4AFC)* übersprungen.
Diese Routine dürfte die einzige sein, die auch für C-Programme in Assembler geschrieben werden muß. Es können noch Fehlerbehandlungs-Routinen eingefügt werden, die eine Plausibilitätsprüfung für D0 durchführen (z.B. D0 gerade, D0>basepage, D0<0x00fffff).

Eine Assembler-Routine kann auch direkt mit dem MONITOR-Befehl gestartet werden. Dabei wird allerdings der Supervisor-Stack benutzt, von dem ich nicht weiß, wieviel Platz dort zur Verfügung steht. Für C-Routinen dürfte der Platz in der Regel nicht ausreichen. Das Programm kann über den C:-Aufruf (*Bild 2*) die Routinen in dem Toolkit benutzen. Hier ist genau eine Routine enthalten, so daß die Übergabe einer Funktionsnummer (hier 0) reiner Luxus ist, aber für Erweiterungen Spielraum läßt. Der mit dem Wert „4" angegebene Offset bewirkt, daß die zu sortierenden Zahlen ab der vierten Stelle aufwärts sortiert werden. Mit einem Offset von „0" erhält man eine komplett sortierte Liste.
Für C-Programmierer ist es einfach, den Sprungverteiler (dispatcher) in C zu schreiben, da die Parameterübergabe den C-Konventionen folgt (*Bild 5*). Das heißt, die Parameter liegen als Wort oder Langwort (L:) auf dem Stapel.

```
dispatcher(nummer,ptr,off)
  int nummer,off;
  char *ptr;
  { switch(nummer)
    {
      case 0: /* shellsort */
              break;
       case 1: /* erweiterung 1 */
               break;
    }
  }
```

Das in Bild 5 angewandte Verfahren erspart die Parameter-Übergabe von der dispatcher-Routine an die untergeordneten Funktionen, also viele überflüssige move-Befehle. Außerdem ist es so einfacher, einer Routine die verschiedensten Parameter zu übergeben; mal Pointer, mal Ints, mal Chars in wechselnder Reihenfolge und verschiedener Anzahl.
Achtung: Der Füllparameter *filler* muß genau so groß sein, wie die Rücksprungadresse des Prozessors, also long beim 68000 und wahlweise long oder int beim 80xxx, je nach Speichermodell. Es ist auch denkbar, daß ein C-Compiler sich nicht an die Parameter-Reihenfolge auf

```
n=20
DIM a$(n)
t1%=TIMER
FOR i%=0 TO n
  a$(i%)=STR$(RND)
NEXT i%
t2%=TIMER
@shellsort(*a$(),4)
t3%=TIMER
FOR i%=0 TO n
  PRINT a$(i%)
NEXT i%
PRINT (t2%-t1%)/200;" s Feldaufbau"
PRINT (t3%-t2%)/200;" s Sortierung"
'
PROCEDURE shellsort(str.arr%,off%)
  off%=off%+1                !mid$ beginnt Zählung bei 1
  SWAP *str.arr%,a$()
  @shell
  SWAP *str.arr%,a$()
RETURN
PROCEDURE shell
  n%=DIM?(a$())
  gap%=n% DIV 2
  WHILE gap%
    i%=gap%
    REPEAT
      j%=i%-gap%
      WHILE j%>=0
        EXIT IF MID$(a$(j%),off%)<=MID$(a$(gap%+j%),off%)
        SWAP a$(j%),a$(j%+gap%)
        SUB j%,gap%
      WEND
      INC i%
    UNTIL i%>=n%
    gap%=gap%/2
  WEND
RETURN
```

Bild 3. Die Sortierroutine als Procedure in GFA-Basic

```
OPEN "O",#1,"MC_SORT.PRG"
READ a%
WHILE a%>=0
  OUT #1,a%
  READ a%
WEND
CLOSE #1
' MCODE \MC_SORT.PRG
DATA 96,26,0,0,1,136,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,96,44,71,70
DATA 65,66,65,83,73,67,46,80,82,71,0,0,0,0,0,0
DATA 0,0,11,77,67,95,83,79,82,84,46,66,65,83,0,0
DATA 0,0,0,0,0,0,0,0,0,79,250,2,92,47,60
DATA 0,0,3,140,72,122,254,198,66,103,63,60,0,74,78,65
DATA 222,252,0,12,72,122,0,76,63,60,0,38,78,78,92,143
DATA 72,122,255,214,72,122,255,188,72,122,255,164,66,103,63,60
DATA 0,75,78,65,222,252,0,14,63,0,72,122,0,56,63,60
DATA 0,38,78,78,92,143,48,31,103,20,72,122,0,218,63,60
DATA 0,9,78,65,92,143,63,60,0,1,78,65,84,143,66,103
DATA 78,65,35,248,0,16,0,0,1,136,33,252,0,0,0,176
DATA 0,16,78,117,33,250,0,222,0,16,78,117,32,64,32,188
DATA 0,0,0,190,84,175,0,2,78,115,48,47,0,4,103,4
DATA 112,255,78,117,32,111,0,6,34,88,12,80,0,1,102,240
DATA 32,25,52,47,0,10,46,0,226,143,103,108,34,7,210,129
DATA 210,135,210,129,44,7,42,6,154,135,107,62,42,69,219,205
DATA 219,197,219,205,219,201,77,245,24,0,38,93,54,29,40,94
DATA 56,30,150,66,152,66,214,194,216,194,83,68,107,24,83,67
DATA 107,24,185,11,103,244,98,18,54,37,58,166,60,131,38,37
DATA 42,166,44,131,96,194,83,67,106,238,82,134,188,128,101,182
DATA 96,166,32,89,50,25,103,16,72,193,82,65,2,65,255,254
DATA 69,233,255,250,33,138,24,0,81,200,255,232,4,128,0,1
DATA 0,0,106,222,78,117,13,10,9,70,101,104,108,101,114,32
DATA 98,101,105,109,32,78,97,99,104,108,97,100,101,110,13,10
DATA 9,69,105,110,101,32,84,97,115,116,101,32,100,114,129,99
DATA 107,101,110,0,0,0,0,154,6,20,0
DATA -1
```

Bild 4. Ladeprogramm der Assembler-Routine aus Bild 1

dem Stack hält. Dann funktionieren solche Tricks natürlich nicht. Solche Tricks sind aber weitverbreitet, so daß ein solcher Compiler bei vielen Programmen Schwierigkeiten hätte.
Die Sortier-Routine ist im Assembler-Programm als einfaches Shell-Sort ausgeführt. Ein GFA-Basic-Listing (*Bild 3*) ist zum Vergleich aufgeführt. In *Bild 4* ist das Assembler-Programm zum Einlesen mit GFA-Basic wiedergegeben. Daß sich die Einbindung von Assembler-Routinen durchaus lohnt, macht ein kleiner Test deutlich: Der GFA-Basic-Interpreter benötigte mit dem Programm aus Bild 3 für das Sortieren von 20 000 zufällig erzeugten Zahlen etwa 14 Minuten. Durch den GFA-Basic-Compiler konnte die Sortierzeit um glatte 10 Minuten verkürzt werden. Die in Bild 1 vorgestellte Assemblerroutine sortierte die 20 000 Zahlen in weniger als 22 Sekunden!

Stringfeldaufbau

Wichtig zum Verständnis der Sortier-Routine ist noch der Aufbau eines GFA-Basic-Stringfeldes:
Der Arraypointer (*a$()) liefert die Adresse, an der der Felddescriptor gespeichert ist. Dieser setzt sich aus einem Zeiger (long) und der Anzahl der Dimensionen (word) zusammen. Eine „1" sagt aus, daß das Feld eindimensional ist. Der Zeiger zeigt auf ein Langwort, das die Anzahl der Feldelemente angibt. Im Anschluß daran sind die einzelnen Strings gespeichert, wieder mit Descriptoren, die sich aus Adresse und Länge des Strings zusammensetzen.
Ein String in GFA-Basic enthält zusätzlich noch einen Zeiger (Backtrailer) hinter dem eigentlichen Stringinhalt, der zurück auf den Descriptor zeigt. Diese Trailer bewirken eine erhebliche Beschleunigung der Aufräumarbeiten (Garbage Collection), wenn der Stringspeicher überläuft (100 000 Strings aufräumen in weniger als einer halben Sekunde). Diese Zeiger liegen übrigens immer auf geraden Adressen, wie auch die Strings selbst. Bei gelöschten Strings wird der Backtrailer durch die negative und begradigte Stringlänge ersetzt.
Die Strings werden zunächst ohne Berücksichtigung dieser Backtrailer sortiert, die erst anschließend in einem Extradurchlauf angepaßt werden. Ohne diese Anpassung würde das GFA-Basic später abstürzen.
Die Aufteilung in Sortierung und Backtrailer-Anpassung ergibt einen erheblichen Geschwindigkeitsgewinn. Einem GFA-Basic-String kann übrigens durch Setzen von Adresse und Länge ein beliebiger Speicherbereich von bis zu 32 KByte mit gerader Startadresse zugewiesen werden. Wichtig ist dabei nur, daß der Backtrailer vorher ordnungsgemäß verändert wird, am einfachsten durch A$="". Dieser String wird dann auch bei einer Garbage-Collection nicht verändert. Vor Veränderungen des Strings durch andere Befehle als MID$()=, LSET oder RSET muß der String dann durch Überschreiben der Länge mit Null definiert gelöscht werden.

Literatur

[1] *Ostrowski, Frank*; Über mein GFA-Basic. GFA-Systemtechnik, Düsseldorf 1987.

[2] *Litzkendorf, Uwe*; Das große GFA-Basic-Buch. Data-Becker, Düsseldorf 1986.


```
#define RETADR long              /* für 68000 */
dispatcher(num)
  int num;
  { switch(num)
    { case 0: shellsort(); break;
      case 1: routine1();  break;
    }
  }

shellsort(filler,dummy,ptr,offset)
  RETADR filler;
  int dummy;
  char *ptr;
  int offset;
  {
    /* sortieren */
  }

routine1(filler,dummy,par1,par2,par3,par4)
  RETADR filler;
  int dummy;
  int par1,par2,par3,par4;
  {
    /* irgendwas */
  }
```

Bild 5. So sieht der Sprungverteiler in C aus

Helmut Heimann

Hercules-Grafiken ausdrucken

Ein speicherresidentes Druckprogramm

Computergrafiken sind nicht ohne weiteres auf einem Drukker auszugeben. Ein Assembler-Programm löst dieses Problem, ohne den Computer während des Druckens zu blockieren.

Vor einigen Jahren setzte Hercules einen Standard bei den Monochrom-Grafikkarten. Hercules-Karten haben eine Auflösung von 720*348 Punkten. Da es sich nicht um ein IBM-Produkt handelt, wird der Ausdruck einer Hercules-Grafik vom Betriebssystem nicht unterstützt. Der Anwender ist gezwungen, sich eine Hardcopy-Routine selbst zu schreiben oder Grafik-Programme zu verwenden, die bereits eine solche Routine enthalten. HGC_BACK ist ein speicherresidentes Hintergrund-Hardcopy-Programm für die Hercules-Karte. Das Programm ist für Epson-Drucker angepaßt. Durch gleichzeitiges Drücken der Shift- und der PrtSc-Tasten wird HGC_Back aktiviert. Zwischen drei Funktionen kann der Anwender auswählen:

- Ausdruck des Bildschirminhalts im Grafik-Modus. Eingabe: H oder h
- Abbruch des gerade laufenden Ausdrucks. Eingabe: Esc-Taste
- Ausdruck einer Textseite. Eingabe: alle Zeichen außer H, h und der Esc-Taste

Das Programm sollte bereits nach dem Booten installiert werden. In die AUTOEXEC.BAT-Datei muß der Anwender folgende Zeile einfügen: HGC_BACK. HGC_BACK sollte möglichst als letztes speicherresidentes Programm geladen werden.

Interrupt 5H geschickt verwendet

Nach dem Laden des Programmes wird die Routine INT5_PRG angesprungen. Dieser Programmteil modifiziert den Treiber des Interrupts 5H. Der neue Treiber prüft, ob er aus einem Turbo-Pascal-Programm mit der Prozedur HGC_BACK (*Bild 1*) oder extern durch die Tastatur aufgerufen wurde.

Erfolgte der Aufruf über die Tastatur, erwartet das Programm über den Interrupt 16H noch ein weiteres Zeichen von der Tastatur, um dann abhängig von diesem die entsprechende Aktion einzuleiten.

Ist eine Hardcopy gewünscht, sichert die Routine HGC_save den aktuellen Bildschirminhalt, setzt die Variable copy_running auf 1 und gibt die Kontrolle an das gerade im Vordergrund laufende Programm zurück. Wird eine Hardcopy der zweiten Grafikseite gewünscht, so muß das Programm nach

```
procedure HGC_BACK;

var register:record ax,bx,cx,dx:integer;
             end;
begin
 register.ax:=$abcd
 intr(5,register)
end;
```

Bild 1. Eine Turbo-Pascal-Prozedur ruft das speicherresidente Druckprogramm auf

```
code          segment
              assume cs:code
anfang:
              jmp      init_hgc_back
;==========================================================================;
; DEKLARARTIONSTEIL
;==========================================================================;
;------------ Steuerzeichen für Drucker
steuerung     db    0ah,0dh,1bh,31h,1bh,'K',144,1
                           ;1bh,31h = ESC 1 für 7/72 Zoll Zeilenabstand
                           ;1bh,'K' = Grafik in einfacher Dichte
              db    0      ;Endezeichen

wait_n_times  equ   125    ;Wartezyklen für "PRINTER not busy"
                           ;(für eigenen Drucker anpassen)
                           ;(Grenzen 80<x<150)

copy_running   dw   0      ;Anzeige, ob Hardcopy gestartet
break          dw   0      ;Anzeige: "Hardcopy nach dieser Zeile abbrechen"
drucker        dw   ?      ;ADRESSE von LPT1

max_hoehe      equ  400    ;Höhe des zu druckenden Bildes
                              ;Für Hercules-Karte werden nur 348 Zeilen bedruckt

max_breite     equ  90     ;Breite in Bytes
spaces         equ  25     ;insgesamt werden 400 Bytes in der Höhe gesendet,
                           ;die ersten 25 enthalten noch keine Bildinform.
X              dw   Offffh ;Zwischenspeicher für aktuelle Koord.
Y              dw   max_hoehe

zwischen       dw   OFFSET steuerung  ;temporäres Register

;==========================================================================;
;          Treiber für Interrupt 5H ( Hardcopy-Interrupt)
int5_prg:
          sti                  ;Interrupts erlauben
          push ax              ;Eingangsparameter retten

;-.mit AX=OABCDh kann die Hardcopy auch durch ein selbstgeschriebenes Programm
;- aktiviert werden
          cmp ax,0abcdh        ;Schlüssel für extern aktivierte Hardcopies
          jz hgc_save

;- Wenn nicht direkt aktiviert, dann wird jetzt Eingabe des Benutzers verlangt
          mov ah,0
          int 16h              ;BIOS-Funktion: Warte auf Tastendruck
          cmp ax,2368h         ;Wurde 'h' gedrückt ?
          jz hgc_save
          cmp ax,2348h         ;Wurde 'H' gedrückt ?
          jz hgc_save
```

Bild 2. Dieses Assembler-Programm gibt den Bildschirminhalt auf den Drucker aus

```
        cmp ax,011bh     ;Wurde <ESC> gedrückt ?
        jz exit_int5_prg ;Mit ESC wird Hardcopy abgebrochen
        pop      ax

alte_funktion:           ;Alten INT 5 anspringen, dh. Text-Hardcopy
            db    0eah
i5_copy_ofs dw     ?     ;Offset des alten INT 5
i5_copy_seg dw     ?     ;Segment des alten INT 5

exit_int5_prg:
        pop ax
        mov cs:break,1   ;Abbruch der Hardcopy nach nächster Zeile
        iret

;---------- Sichern des Grafikpuffers
HGC_save:
        cmp cs:copy_running,1  ;Wird noch gedruckt ?
        jnz save_gr_seite      ;nein, dann aktuelle Grafikseite sichern
        pop ax
        jmp end_save
save_gr_seite:
        push ds
        push es
        push si
        push di
        push cx
;-------- HGC-Grafikbildschirm sichern
        mov ax,0b000h
;-- Wird eine Hardcopy der zweiten Grafikseite gewünscht, so ist statt b000
;-- einfach b800 zu schreiben
        mov ds,ax
        xor si,si              ;Quelle= 0B000h:0000  /  0B800h:0000 für Seite 2
        push cs
        pop es
        lea di,Screen_save     ;Ziel  =   CS:Screen_save
        mov     cx,8000h       ;Anzahl der zu übertragenden Worte
                               ;(insges. 32 kb)
        rep movsb              ;Übertragung durchführen
        mov ax,1
        mov cs:copy_running,ax ;Anzeigen, daß Hardcopy läuft
        pop cx
        pop di
        pop si
        pop es
        pop ds
        pop ax                 ;Das AX vom Aufruf muß noch vom
                               ;Stack entfernt werden
end_save:
        mov cs:break,0         ;BREAK-Flag zurucksetzen
        iret                   ;Nach Aktivierung das Programm verlassen

;============ Spooler =====================================================;
;----- eigentlicher Druckerspooler

spooler:
        push ax                 ;benutzte Register sichern
        push bx
        push cx
        push dx
        push ds
        mov ax,cs:copy_running  ;läuft Hardcopy ?
        cmp ax,1
        jnz exit_spooler        ;nein, dann alte
                                ;Timerroutine anspringen

;---- weiter kopieren
        mov cx,15               ;Versuche 15 Bytes pro Aufruf
                                ;zu senden
follow_char:
        mov bx,wait_n_times     ;Warte auf den Drucker
pr_ready:
        call drucker_ready      ;Ist Drucker bereit ?
                                ;(AH=0 heißt:Drucker bereit)
        cmp ah,0
        jz hole_char
        dec bx
        jnz pr_ready
        jmp exit_spooler        ;Time-Out
hole_char:
        push cx
        call char_from_HGC      ;nächstes zu druckendes Zeichen
        pop cx
        jnc drucke
;-------- kein Zeichen mehr im Puffer
        mov cs:copy_running,0   ;ja, dann Hardcopy beendet
        mov cs:break,0          ;BREAK aufheben
        mov al,0ah
        call dru_char
        mov al,0dh
        call dru_char           ;Leerzeile drucken
        jmp exit_spooler

drucke:
        push cx
        call dru_char           ;Sonst drucke das Zeichen in AL
        pop cx
        loop follow_char        ;Anzahl der zu sendenden Zeichen pro
                                ;Interrupt erreicht ?
exit_spooler:

        pop ds
        pop dx
        pop cx
        pop bx
        pop ax

            db    0eah          ;MNEMONIC für JMP FAR
i1c_copy_ofs dw    ?
i1c_copy_seg dw    ?

;-----------------------------------------------------------------------
;                Unterprogramme
;-----------------------------------------------------------------------

drucker_ready:
        mov dx,cs:drucker
        inc dx
        in al,dx                ;Lies Drucker-Status
        and al,098h
        cmp al,10011000b        ;Ist Drucker bereit ?
        mov ah,0
        jz pr_not_busy          ;ja, dann AH=0
        mov ah,0ffh
pr_not_busy:
        ret

dru_char:                       ;Drucke das Zeichen in AL
        mov dx,cs:drucker
        out dx,al               ;Zeichen in den Output-Port
        add dx,2
        mov al,0dh              ;Strobe aktivieren
        out dx,al
        nop                     ;verzögern Centronics-Spezifikation)
        nop
        nop
        mov al,0ch              ;Strobe inaktiv
        out dx,al
        ret

char_from_HGC:
```

```
;------------------------- HGC-Grafikpufferleseroutine------------------------
        mov ax,cs:Y
        cmp ax,max_hoehe            ;Steuerzeichen erforderlich ?
        jc read_screen
        cmp cs:break,1              ;Soll Hardcopy abgebrochen werden ?
        jz  end_hgc_hard
        mov ax,cs:X
        cmp ax,max_breite-1         ;letzte Spalte gedruckt ?
        jnz steuerzeichen
end_hgc_hard:
                                    ;ja, dann ist Hardcopy beendet
; Voreinstellungen für nächste Hardcopy
        mov bx,0ffffh               ;-1, ein noch ungültiger Wert, um das
                                    ;Senden der Steuerzeichen einzuleiten
        mov cs:X,bx
        mov bx,max_hoehe
        mov cs:Y,bx
        stc                         ;gesetztes Carry als Zeichen für das Ende
                                    ;der Hardcopy
        jmp end_hgc_read

steuerzeichen:
        mov bx,cs:zwischen          ;Zwischengespeicherter Stand der
                                    ;Steuerzeichensendung
        mov al,cs:[bx]
        inc bx
        mov cs:zwischen,bx          ;Pointer aktualisieren

        cmp byte ptr cs:[bx],0      ;War dies das letzte Steuerzeichen
                                    ;d.h. wäre das nächste eine Null ?
        jnz noch_eines

        mov bx,OFFSET steuerung     ;ja, dann nächste Stuersequenz vorbereiten
        mov cs:zwischen,bx
        mov bx,max_hoehe-1
        mov cs:Y,bx
        inc cs:X

noch_eines:
        clc
        jmp end_hgc_read

read_screen:                        ;Zeichen muß aus Bildschirmspeicher
                                    ;gelesen werden
;----- einmitten des Bildes auf dem Blatt
        mov bx,cs:Y
        sub bx,spaces               ;Einmitten der Hardcopy durch Leerzeilen
        jc sende_space
        cmp bx,348                  ;Ist max. HGC-Auflösung erreicht ?
        jc sende_info               ;Nein, dann Pixel senden
sende_space:
        xor ax,ax                   ;ja, dann keinen Punkt mehr setzen
        dec cs:Y                    ;aber Y-Koordinate decrementieren
        jmp en_hgc_read             ;und "keinen Fehler" anzeigen
sende_info:
        mov dx,bx                   ;Y sichern
        and bx,03h
        mov cl,13
        shl bx,cl
        shr dx,1
        and dx,0ffeh
        mov ax,dx
        shl dx,1
        add ax,dx
        mov dx,ax
        mov cl,4
        shl dx,cl
        sub dx,ax                    ;Byteadresse berechnen aus:
                                    ; 2000h*(y mod 4)+90*(y div 4)
;-- schnellster Weg, die Byteadresse zu berechnen (aus Turbo-Grafik-Toolbox)
        add bx,dx
        add bx,cs:X                 ;plus Anzahl der Bytes in X-Richtung
        add bx,OFFSET Screen_save   ;Adresse im Puffer
        mov al,cs:[bx]              ;Grafikbits lesen
        dec cs:Y                    ;Y decrementieren
        clc
end_hgc_read:
        ret
;--------------- Puffer für Graphikbildschirm --------------------------
Screen_save     db          7f20h dup(?)

;Puffer so wählen, daß Installationsroutine überschrieben wird, da sie
; nach dem ersten Aufruf sowieso nicht mehr benötigt wird.
;======================================================================;
;-------- Interrupts sollen auf eigene Routinen zeigen
init_hgc_back:
        push ds
        push es
        mov ax,40h
        mov ds,ax                   ;DS auf 40h
        mov bx,8
        mov ax,[bx]                 ;I/O-Adresse des Druckerports holen
        mov cs:drucker,ax           ;und sichern
;-------- INT 5 verbiegen
        mov ax,3505h                ;INT 5h Adresse lesen
        int 21h                     ;Ergebnis in ES:BX
        mov cs:i5_copy_seg,es
        mov cs:i5_copy_ofs,bx       ;Speichern von SEGMENT und OFFSET
        push cs
        pop ds
        cli
        mov dx,OFFSET int5_prg
        mov ax,2505h                ;Vektor auf eigene Routine verbiegen
        int 21h
;-------- INT 1C verbiegen
        mov ax,351Ch                ;INT 1C Adresse lesen
        int 21h
        mov cs:i1c_copy_seg,es      ;und sichern
        mov cs:i1c_copy_ofs,bx
        push cs
        pop ds
        cli
        mov dx,OFFSET spooler
        mov ax,251Ch
        int 21h                     ;Vektor auf eigene Routine verbiegen
;-------- Textausgabe
        push cs
        pop ds
        mov dx,OFFSET ausgabe
        mov ah,9
        int 21h                     ;DOS-Aufruf: Gib String aus
        pop es
        pop ds
        mov dx,init_ende-anfang+100h
        mov cl,4
        shr dx,cl
        mov ax,3100h                ;DOS Aufruf:Resident machen von EXE-Files
        cli
        int 21h                     ;Programm resident machen

ausgabe db      'HGC_BACK installiert !',0dh,0ah,0dh,0ah
        db      'Start mit <SHIFT> <PRTSCR> und',0dh,0ah,0dh,0ah
        db      '      <H> für HGC-Hintergrund-Hardcopies',0dh,0ah,0dh,0ah
        db      'Abbruch mit <SHIFT> <PRTSCR> plus <ESC>$'
;------Installation beendet-----------------------------------------------
init_ende:
code    ends
        end
```

der Marke save_gr_seite geändert werden. Die Anweisung mov ax,0b00h wird durch mov ax,0b800h ersetzt.

Drucker-Spooler

Die eigentliche Grafikausgabe-Routine ist interruptgesteuert. Sie verwendet den Zeitgeber-Interrupt 1CH, den die Hardware alle 54,9 ms auslöst. Die Routine stellt zuerst fest, ob gespoolt werden muß. Dazu wird die Variable copy_running auf eine logische 1 abgefragt. Soll der Bildschirminhalt ausgedruckt werden, prüft die Routine in einer Schleife, ob der Drucker bereit ist, ein Zeichen zu übernehmen. Diese Warteschleife wird abgebrochen, wenn nach 125 Abfragen der Drucker immer noch nicht bereit ist. Sobald der Drucker ein Zeichen übernehmen kann, wird die Routine char_from_HGC aufgerufen. Diese Routine entscheidet, ob ein Druckersteuerzeichen oder ein Byte aus dem Grafikpuffer an den Drucker gesendet werden soll. Das Programm kann für andere Drucker angepaßt werden, da nur die Druckersteuerzeichen für den Zeilenabstand, für die Grafik in einfacher Dichte und für die Zeilenbreite geändert werden müssen.

Als Zeilenbreite wurde 400 gewählt, um das Bild auf dem Blatt zu zentrieren. Die Konstante SPACES gibt die Anzahl der Leerzeichen vom linken Blattrand an. Der Wert dieser Konstanten darf im Bereich zwischen 0 und 52 liegen.
Mit dem MASM (Microsoft Macroassembler) wird das Programm (*Bild 2*) assembliert. Eine EXE-Datei erzeugt der folgende Befehl: LINK HGC_BACK.OBJ.

Handlich und schnell: Texteditor ohne Schnickschnack

Edi heißt ein Texteditor für MS-DOS-Computer, den die Firma Shamrock Software anbietet. So handlich und kurz wie sein Name ist auch das Programm selbst. Es belegt nicht einmal 5 KByte und bietet sich deshalb für all jene an, die mit Disketten arbeiten und darauf neben den Daten ohne großen Platzverlust auch noch einen Editor unterbringen wollen.
Wer nun aber glaubt, wegen seiner Kürze könne das Programm notgedrungen nicht besonders viel leisten, der irrt. Edi kennt sämtliche Funktionen, die man bei der Bearbeitung von Texten häufig braucht. Verzichten muß man vorwiegend auf Eigenschaften, die nur in Spezialfällen benötigt werden. Nun ist es immer eine Frage der persönlichen Erfordernisse, worauf man besonderen Wert legt. Im *Bild* sind deshalb sämtliche Funktionen von Edi aufgelistet. Doch diese Liste allein ist noch kein zuverlässiges Kriterium dafür, wie das Programm im Vergleich zu anderen abschneidet. Ein wesentliches Plus von Edi kann man daraus nämlich nicht ersehen: seine Geschwindigkeit.

Ganz schön schnell

Wer sich jemals mit Textverarbeitungsprogrammen herumgequält hat, die bei längeren Dateien nur noch „dahinschleichen", der wird mit Erstaunen feststellen, wie schnell Edi in der Lage ist, zwischen Textanfang- und -ende hin und her zu springen oder den Text auf dem Bildschirm zu verschieben. Ebenso schnell ist der Editor selbst geladen – dank seiner Kürze. Diese enorme Geschwindigkeit kommt daher, daß das Programm vollständig in Assembler geschrieben ist und – wie schon angedeutet – auf selten benutzte Funktionen verzichtet.
Zu bedienen ist Edi ausschließlich mit den Cursor-Steuertasten (einschließlich Delete, Backspace und Insert) und mit den Funktionstasten des PC. Ob das ein Vorteil oder ein Nachteil ist, ist wieder eine Frage des persönlichen Geschmacks (am einfachsten ist immer das Programm zu bedienen, das man gewöhnt ist). Technisch notwendig ist diese Art der Bedienung bei Edi aus folgendem Grund: Das Programm erlaubt nämlich die Eingabe von Steuerzeichen (Kombinationen mit „Ctrl" oder Escape-Sequenzen) direkt von der Tastatur aus. Dadurch läßt sich jeder Drucker mit all seinen Möglichkeiten betreiben. Ein weiterer Vorteil: Dateien, die Steuerzeichen enthalten, lassen sich problemlos bearbeiten. Die meisten mc-Leser werden abschätzen können, was es bedeutet, wenn man z. B. die Funktion „Suchen und ersetzen" auch mit Steuerzeichen verwenden kann. Unschön, aber nicht zu vermeiden: Die Steuerzeichen werden wie normale Buchstaben behandelt und bei der Formatierung mitgezählt. Zu Edi gehören mehrere Dienstprogramme, unter anderem SETPRINT, LPTGER und CALC. Die ersten beiden beziehen sich auf den Drucker. SETPRINT erlaubt es (auch wenn Edi nicht in Betrieb ist), den angeschlossenen Drucker über die Tastatur auf bestimmte Betriebsarten einzustellen. LPTGER macht ältere Modelle IBM-tauglich, indem es den Zeichensatz umcodiert. Die deutschen Sonderzeichen werden dabei berücksichtigt. Das dritte Dienstprogramm, CALC, stellt einen „Taschenrechner" zur Verfügung, der sich in beliebige Programme einblenden läßt. Er ist insbesondere für Programmierer sehr nützlich, da er sich im Dual-, Oktal-, Dezimal- und Sedezimal-(Hexadezimal-)System betreiben läßt. Außerdem sind auch logische Verknüpfungen der Operatoren möglich. Alles in allem: Edi kann sich sehen lassen – nicht nur, wenn man den günstigen Preis (98 DM) berücksichtigt. *Ho*

Taste	normal
F1	Hilfsmenü 1
F2	Dateinamen eingeben
F3	Datei laden
F4	Datei sichern
F5	Wort suchen
F6	Weitersuchen
F7	Blockanfang markieren
F8	Blockende markieren
F9	Zeile umbrechen
F10	Beenden

Taste	mit Shift
F1	Hilfsmenü 2
F2	Datei anhängen
F3	Directory wechseln
F4	Directory anzeigen
F5	Zeile löschen
F6	Disk-Platz
F7	Block löschen
F8	Block kopieren
F9	Marken löschen
F10	Rechter Rand

Taste	mit Alt
F1	Hilfsmenü 3
F2	Freier Platz/Zeilennr.
F3	Nach Zeilennr.
F4	Block sichern
F5	Ersatzwort eingeben
F6	Wort ersetzen
F7	Zum Blockanfang
F8	Zum Blockende
F9	Videosynch. ein/aus
F10	Suchmodus umschalten

Taste	mit Ctrl
F1	Hilfsmenü 4
F5	Wort links
F6	Wort rechts
F8	Block verschieben
<-	Zeilenanfang
->	Zeilenende

Diese Funktionen kennt Edi

Joachim Lange

Software für das MIDI-Interface

Der Apple-II übernimmt jetzt auch die Sound-Verwaltung

Die MIDI-Schnittstelle eröffnet dem Musiker zahlreiche Möglichkeiten, elektronische Musikinstrumente anzusteuern. Nachdem für das Apple-MIDI-Interface bereits ein Sequencer-Programm vorgestellt wurde [1], beschäftigt sich dieser Beitrag mit der Bedienung eines MIDI-Gerätes über die Schnittstelle.

Wer einen programmierbaren Synthesizer sein eigen nennt, wird sich sicher schon an verschiedenen Klangschöpfungen versucht haben. Nach einigen Experimenten wird er sich vielleicht gefragt haben, wo er seine Kreationen wohl unterbringen (sprich konservieren) soll. Die mühsam über mehrere Unterebenen (Mehrfachtastenbelegung) eingegebenen Sound-Daten möchte er ja nicht bei nächster Gelegenheit wieder von neuem eintippen, und der variable Speicher ist außerdem bereits gefüllt mit Sounds, die man aus dem Festwertspeicher kopiert hat.

Die dem Synthesizer mitgelieferten Sound-„Merkblätter“, auf denen man die Klangparameter eintragen kann, vermitteln auch nicht das Gefühl von Datensicherheit und Arbeitserleichterung. Zum Glück kann man die Daten auf einer Kassette speichern und sie wieder in den Synthesizer laden. Sie müssen allerdings gut Buch führen um zu wissen, auf welcher Kassette welche Daten sind.

Nun, wenn man schon einen Computer mit einer schnellen MIDI-Schnittstelle hat, kann man ihm auch Klanginformationen zum Speichern übermitteln. Das kann natürlich nur funktionieren, wenn der Synthesizer entsprechend ausgestattet ist. Die Geräte der Yamaha-DX-Serie (es gibt natürlich auch andere Hersteller) erlauben diese Art der Sound-Datensicherung. Der Bedienungskomfort reicht sogar so weit, daß praktisch alles, was man am Synthesizer per Hand einstellen kann, auch über die MIDI-Schnittstelle einstellen kann. Inwieweit dies sinnvoll ist, ist eine ganz andere Frage.

MIDI-Fernsteuerung

Um die Steuerung eines Gerätes über die Schnittstelle zu verstehen, müssen wir uns etwas genauer mit dem allgemeingültigen MIDI-Protokoll befassen [3]. Im Datensatz für MIDI-Statusbytes ($80 bis $FF) sind einige Bytes (der Bereich $F0...$FE) für System-Informationen reserviert. Speziell das Byte $F0 kündigt an, daß SYSTEM-EXCLUSIVE-Daten gesendet werden. Damit gibt man dem Benutzer die Freiheit, beliebige Daten zu übertragen, die außerhalb der Norm liegen. Das Datenformat für diesen speziellen Fall ist folgendermaßen definiert:

F0 ii d1 d2 d3 d4 ..... dn F7 (Hex)

Das Byte $F0 kündigt SYSTEM-EXCLUSIVE-Daten an, gefolgt von einem herstellerspezifischen Identifikationsbyte und beliebig vielen Datenbytes d1...dn. Das Ende des Datenstroms wird durch das Byte $F7 (EOX = END OF EXCLUSIVE) signalisiert. Was zwischen $F0 und $F7 übertragen wird, bleibt also ganz dem Benutzer bzw. dem Gerätehersteller überlassen. Die einzige Beschränkung liegt im Wertebereich; es dürfen keine Statusbytes (Bit 7 = 1) in den Daten d1...dn enthalten sein. Das ist die Grundlage für den hier beschriebenen Datenaustausch.

Wir sind hier bereits an einem Punkt angelangt, an dem man nicht weiter allgemeingültig für alle MIDI-Geräte sprechen kann. Sicher sind nicht alle MIDI-Synthesizer mit den genannten Eigenschaften ausgestattet, und bei denen, die es können, verlangt jeder nach unterschiedlichen Anweisungen. Hier hilft nur das Vertiefen in das (hoffentlich ausführliche) Handbuch des eigenen Synthesizers. Das in diesem Beitrag vorgestellte Programm ist weitgehend allgemeingültig gehalten und so flexibel aufgebaut, daß dem Leser die Anpassung an das eigene System sicher nicht schwerfällt. Die Ausführungen im Handbuch sind für den Programmierer oft recht knapp gehalten, und so lohnt es sich, die Verhältnisse etwas detaillierter zu betrachten.

Im folgenden soll als Beispiel der Datenaustausch mit dem Yamaha-Synthesizer DX-27 [2] beschrieben werden. Die MIDI-Befehlssequenzen und das Sound-Datenformat sind auch uneingeschränkt für den softwarekompatiblen DX-100 und den DX-21 gültig. Die Sounds können bei diesen drei Synthesizern untereinander ausgetauscht werden.

Zum besseren Verständnis hier noch ein paar Worte zu den Speicherverhältnissen im DX-27. Der Synthesizer besitzt ein ROM mit 2 * 4 * 24 vorprogrammierten Klängen und einen CMOS-RAM-Bereich (Internal-Bank genannt) für 24 frei programmierbare Klänge. Zum Editieren des einzelnen Sounds gibt es noch den sogenannten Arbeitsspeicher. Speziell die Internal-Bank und der Arbeitsspeicher interessieren uns für den Datenaustausch.

Daten abzapfen

Wir setzen uns zum Ziel, die im CMOS-RAM des Synthesizers gespeicherten Sound-Daten in den Speicher des Apple zu holen und auf Diskette zu schreiben. Dazu gibt es mehrere Möglichkeiten. Die erste läßt sich bereits mit dem Sequencer-Programm aus [1] ausführen.

Man braucht den Sequencer nur auf „RECEIVE“ zu stellen und am Synthesizer mit der Funktionstastenfolge

"FUNCTION", "SYS-INFO", "SYS-INFO", "YES"

bewirken, daß dieser seine Klangdaten (hier die ganze Sound-Bank) ausgibt.

Wenn der Synthesizer nach etwa drei Sekunden wieder seine Betriebsbereitschaft anzeigt, haben wir die Daten bereits im Speicher stehen. Wir können dann den Receive-Modus abbrechen und die Daten auf Diskette schreiben. Das ist bereits viel komfortabler und schneller als eine Kassettenaufzeichnung, und wir können den gesicherten Daten einen Dateinamen zuordnen.
Es geht jedoch noch eleganter: Die übrigen Möglichkeiten bestehen darin, den Synthesizer über die Schnittstelle anzuweisen, seine Daten auszugeben. Man kann entweder die Handbetätigung der Funktionstasten softwaremäßig simulieren oder dem Synthesizer spezielle Ausgabeanweisungen übermitteln. Die in diesem Zusammenhang gültigen Befehlssequenzen sind in *Bild 1* zusammengestellt [2].

```
a) Softwaremäßiges Betätigen der Funktionstasten:

FO  43  10  08  sn  dd  F7    (Hex)

$FO:  SYSTEM EXCLUSIVE
$43:  Identifikationsbyte (Yamaha)
$10:  Substatus: Parameteränderung
$08:  Parametergruppe: Tastenbetätigung
 sn:  Schalter-Nummer
 dd:  Datum  (dd=0: aus,  sonst: ein)
$F7:  END OF EXCLUSIVE

b) Direkte Parameteränderung (zum Verändern von
   beliebigen Einstellungen, zum Editieren von Sounds
   und für die Ausgabeanforderung):

FO  43  10  12  pp  dd  F7    (Hex)

$FO:  SYSTEM EXCLUSIVE
$43:  Identifikationsbyte (Yamaha)
$10:  Substatus: Parameteränderung
$12:  Parametergruppe: direkte Parameteränderung
 pp:  Parameter-Nummer (für Sound-Bank ausgeben: $6E)
 dd:  Parameter-Datum ($00...$7F, hier $01)
$F7:  END OF EXCLUSIVE

c) Spezielle Ausgabeanforderung für Sound-Daten:

FO  43  20  ff  F7    (Hex)

$FO:  SYSTEM EXCLUSIVE
$43:  Identifikationsbyte (Yamaha)
$20:  Substatus: Ausgabeanforderung
 ff:  gewünschtes Format
        ff=$03:  1 Sound
        ff=$04:  Sound-Bank
$F7:  END OF EXCLUSIVE
```

Bild 1. Die gültigen SYSTEM-EXCLUSIVE-BEFEHLE zum Anfordern von Sound-Daten beim DX-27

Die unter c) angegebene Befehlssequenz ist am einfachsten und kompaktesten. Man kommt mit fünf Bytes aus, während man mit der softwaregesteuerten Tastenbedienung nach a) meist mehrere Befehlssequenzen absetzen muß. Wie aus der Aufstellung c) ersichtlich ist, gibt es für die Übertragung der Sound-Daten zwei verschiedene Formate. Man kann grundsätzlich die ganze Sound-Bank, bestehend aus 32 Einzel-Sounds, oder einen beliebigen einzelnen Sound vom Synthesizer abrufen. Es besteht jedoch ein Unterschied in der Datenstruktur zwischen gesammelten Bank-Sounds und den Einzel-Sounds. In den Einzel-Sounds sind alle Sound-Parameter (Klangeigenschaften) einzeln in Bytes codiert, während in den einzelnen Blökken der Sound-Bank mehrere Sound-Parameter mit kleinem Wertebereich in einem Byte zusammen codiert sind (vermutlich, um intern ROM- und RAM-Speicherplatz zu sparen). Zum Beispiel sind die vier Parameter

7 Envelope-Generator-Vorspannungsansprache
8 Amplitudenmodulation ein/aus
9 Anschlagdynamik
10 Ausgangspegel

im Parameter Nummer 6 des jeweiligen Bank-Soundblocks zusammengefaßt, während sie im Datenblock der Einzel-Sounds getrennt vorliegen. Daher belegt ein komprimierter Sound in der Sound-Bank 73 Bytes, als einzelner Sound im Arbeitsspeicher jedoch 93 Bytes. Ein einzelner Sound wird beim Datenaustausch zwischen Synthesizer und Computer immer aus dem Arbeitsspeicher (mit dem gespielt oder editiert wird) gelesen bzw. in diesen hineingeschrieben. Die Daten müssen daher nach der Übermittlung vom Computer in einen der 24 Bank-Speicher des CMOS-RAMs kopiert werden, um sie permanent zu speichern. Der Datenblock für Einzel-Sounds kann zum Beispiel zum Editieren vom Computer aus dienen. Außerdem kann man damit einzelne Sounds innerhalb der Internal-Bank umgruppieren. Das alles mag für manchen Leser etwas verwirrend klingen, wer jedoch schon Programmiererfahrung mit Synthesizern hat, wird verstehen, wovon die Rede ist. Nach diesem kleinen Ausflug zurück zum Datenaustausch.

Nachdem wir dem Synthesizer eine Ausgabeanforderung geschickt haben, gibt dieser seine Sound-Daten aus. Nun brauchen wir nur noch dafür zu sorgen, daß wir auch alle Daten mit dem Apple empfangen. Dazu müssen wir eine schnelle Empfangsroutine benutzen, die laufend die Register des ACIA abfragt und die eintreffenden Daten nacheinander im Apple-Speicher ablegt, damit wir sie später auf Diskette sichern können. Damit haben wir die gewünschten Daten „im Kasten". Jetzt fehlt uns noch der umgekehrte Weg der Sound-Übertragung.

Sound-Daten senden

Schauen wir uns einmal die vom DX-27 empfangenen Daten näher an: Der Datenblock für einen Einzel-Sound ist in *Bild 2* dargestellt. Der Synthesizer sendet nicht die reinen Datenblöcke, bestehend aus den Klangparametern, sondern einen kompletten Datensatz. Die ersten sechs Bytes bilden einen Vorspann, der einem anderen MIDI-Gerät mitteilt, daß bestimmte Daten im Anmarsch sind. Er besteht zunächst aus dem bereits erwähnten Statusbyte und dem Identifikationsbyte. Danach folgen zwei Bytes für Substatus und Datenformat, weitere zwei geben die Nettodatenlänge an. Erst dann folgen die Klangparameter-Daten. Die Bedeutung der einzelnen Bytes ist im Handbuch [2] genau beschrieben, wir wollen hier nicht näher darauf eingehen.

Die letzten beiden Bytes des Datensatzes gehören nicht zu den Nettodaten, sie enthalten eine Prüfsumme und die END-OF-EXCLUSIVE-Nachricht. Der Vorspann für eine Sound-Bank ist genauso aufgebaut, das vierte Byte hat jedoch den Wert $04 (anderes Sound-Format). Der Datensatz ist also derart gestaltet, daß er für einen gleichartigen, zweiten Synthesizer unmittelbar zum Empfangen geeig-

```
]BLOAD CELESTE,A$1000

]CALL-151

*1000.1064

1000- F0 43 00 03 00 5D 19 1C
1008- 00 06 01 00 01 00 00 00
1010- 45 07 00 1F 10 00 06 01
1018- 00 01 00 00 00 34 16 06
1020- 1F 0A 00 06 01 00 01 00
1028- 00 00 63 04 06 1F 0D 00
1030- 04 01 00 00 00 00 00 60
1038- 04 00 04 00 19 00 00 00
1040- 00 02 05 00 24 00 00 00
1048- 00 63 01 00 00 32 00 32
1050- 00 32 00 43 65 6C 65 73
1058- 74 65 20 20 20 63 63 63
1060- 32 32 32 17 F7
```

Bild 2. Der Datenblock eines Einzel-Sounds, wie er vom Synthesizer gesendet wird: Die ersten sechs Bytes bilden den Vorspann; unter den Adressen $1053...$105C ist der Name gespeichert; die letzten beiden Bytes sind Prüfsumme und EOX

net ist. Damit erübrigt sich eine besondere Empfangsanweisung zum Übermitteln von Daten. Man kann die mit dem Computer bereits empfangenen Sound-Daten einfach dem Synthesizer zurücksenden. Nebenbei sei noch bemerkt, daß der Vorspann eines Sound-Datensatzes analog zu den in Bild 1a und 1b gezeigten Befehlssequenzen aufgebaut ist, weshalb er auch als Empfangsanweisung für den Synthesizer aufgefaßt werden kann.

Anpassen an das eigene System

Nach der vorangegangenen Beschreibung dürfte es ein leichtes sein, für die korrekte Bedienung des eigenen Synthesizers die entsprechenden Befehlssequenzen im Programm vorzusehen. Um eine bestimmte Befehlssequenz über die MIDI-Schnittstelle auszugeben, bedarf es lediglich der folgenden Zeilen:

```
MIDISEND JSR  MIDIOUT
         DFB  aa
         DFB  bb
         DFB  cc
          :
          :
         DFB  $FF ;Datenende
CONTINUE ...
         ...  (weiteres Assembler-
               programm)
```

Darin ist die aufgerufene Subroutine MIDIOUT die in *Bild 3* enthaltene universelle Ausgaberoutine. Alle diesem Aufruf folgenden Bytes werden bis zu einem $FF (ohne dieses Byte) an die Schnittstelle ausgegeben. Das Byte $FF ist unbedingt vorzusehen, da sonst das folgende Programm nicht korrekt ausgeführt wird. Dies ist der einzige gerätespezifische Programmteil, der für das Übermitteln einer Befehlssequenz an ein MIDI-Gerät notwendig ist. Allerdings muß man für eine vollständige Bedienung mehrere solcher Befehlssequenzen im Programm vorsehen.

Das Programm

Mit dem Programm zur Sound-Verwaltung in Bild 3 kann man sich in Verbindung mit dem MIDI-Interface [1] Sound-Daten vom Synthesizer in den Apple-Speicher holen und auf Diskette sichern. Bei Bedarf kann man den gesamten Vorgang umgekehrt vollziehen und damit den Synthesizer auf schnelle Art mit neuen Sounds „beschicken". Der Datenaustausch für eine ganze Sound-Bank dauert nur etwa drei Sekunden. Auf einer einzigen Diskette lassen sich mindestens 25 ganze Sound-Bänke speichern. Das Programm ist der Einfachheit halber für einen bestimmten Slot geschrieben. Außerdem ist es mit der dadurch möglichen direkten Adressierung ein wenig schneller als mit der indizierten. Das MIDI-Interface muß in Slot 4 stecken; durch Ändern des Einstellparameters SLOT kann das Programm leicht für einen anderen Slot assembliert werden. Die Befehlssequenzen gelten für die Yamaha-Synthesizer DX-27, DX-21 und DX-100, sie sind für andere Geräte entsprechend zu ersetzen. Das Programm startet man einfach mit BRUN SOUNDVERW, es meldet sich dann mit seinem Menü am Bildschirm. Im einzelnen stehen folgende Kommandos zur Auswahl:

1: Einen Sound vom Keyboard holen

Es wird eine Ausgabeanforderung für einen Sound nach Bild 1c an den Synthesizer ausgegeben. Daraufhin wartet das Programm auf eintreffende Daten und legt sie im Apple-Speicher ab Adresse $1000 ab. Um den Computer bei fehlendem Datenstrom nicht endlos warten zu lassen, fragt das Programm laufend die Tastatur ab und bricht ab, wenn eine Taste gedrückt wurde. Damit bei einer Fehlfunktion nicht gleich der DOS-Speicherbereich überschrieben wird, wird auch laufend geprüft, ob die (willkürlich angenommene) Speicheradresse $9000 überschritten wird und gegebenenfalls abgebrochen. Die Längeninformation im Menü zeigt an, ob korrekt empfangen wurde. Das wird angenommen, wenn ein EOX ($F7) empfangen wird. Im Fehlerfall ist die Datenlänge null. Ein Prüfsummentest der Übertragung wäre möglich, ist aber nicht vorgesehen, da die Kommunikation erfahrungsgemäß reibungslos funktioniert. Der Datenblock ist beim DX-27 $65 Bytes lang, wenn er richtig empfangen wurde (101 Bytes zusammengesetzt aus 6 Vorspann-, 93 Nettodaten-, 1 Prüfsumme-, 1 EOX-Bytes). Welchen Sound man abruft, ist von der Einstellung am Synthesizer abhängig, es

```
1    ****************************************
2    *                                      *
3    *   S O U N D V E R W A L T U N G      *
4    *                                      *
5    *   FUER APPLE-II UND SYNTHESIZER      *
6    *      YAMAHA  DX-27 / DX-100          *
7    *                                      *
8    *        VON  JOACHIM LANGE            *
9    *                                      *
10   ****************************************
11
12   *   EINSTELLPARAMETER
13   *
14   SLOT      EQU  $40
15   BUFBEGIN  EQU  $1000
16   *
17   *   ZERO-PAGE-ADRESSEN
18   *
19   VERTCURS  EQU  $25
20   PROMPT    EQU  $33
21   LINECNT   EQU  $E0
22   TEMP      EQU  $E2
23   LENGTH    EQU  $EA
24   PTR       EQU  $EC
25   PTR2      EQU  $EE
26   INPTR     EQU  $FA
27   OUTPTR    EQU  $FC
28   *
29   *   GENERAL  EQUATES
30   *
31   INBUFF    EQU  $200
32   KEYBD     EQU  $C000
33   KEYBDSTB  EQU  $C010
```

```
34   *
35   *    ACIA-ADRESSEN
36   *
37   TDATREG   EQU  $C081+SLOT
38   RDATREG   EQU  $C081+SLOT
39   CTRLREG   EQU  $C080+SLOT
40   STATREG   EQU  $C080+SLOT
41   *
42   *    ROM - ROUTINEN
43   *
44   GETLN     EQU  $FD6A
45   GETCHAR   EQU  $FD0C
46   PRBL2     EQU  $F94A
47   COUT      EQU  $FDED
48   COUT1     EQU  $FDF0
49   CROUT     EQU  $FD8E
50   PRERR     EQU  $FF2D
51   PRBYTE    EQU  $FDDA
52   PRNTAX    EQU  $F941
53   HOME      EQU  $FC58
54   VTABZ     EQU  $FC24
55   *
56   *    DOS-ADRESSEN
57   *
58   AMPERSND  EQU  $3F5
59   DOSCOLD   EQU  $3D3
60   ERRVEC    EQU  $9D5A
61   LENGTADR  EQU  $AA60
62
63   **************************
64   *
65   *    INITIALISIERUNG
66   *
```

Bild 3. Das Programm zur Sound-Verwaltung für den Apple-II (diese Routine wird in ähnlicher Form zur Ausgabe von Zeichenketten verwendet, das Original stammt von Andy Hertzfeld)

```
                 67  ********************************
                 68
                 70            ORG   $900
                 71
0900: A9 B1      72  START     LDA   #<ERRSUB     ;DOS-ERROR-
0902: A0 0D      73            LDY   #>ERRSUB     ; VEKTOR
0904: 8D 5A 9D   74            STA   ERRVEC       ; SETZEN
0907: 8C 5B 9D   75            STY   ERRVEC+1
090A: A9 00      76            LDA   #<START      ;AMPERSAND-
090C: A0 09      77            LDY   #>START      ; VEKTOR
090E: 8D F6 03   78            STA   AMPERSND+1   ; SETZEN
0911: 8C F7 03   79            STY   AMPERSND+2
0914: A9 00      80            LDA   #$00         ; BASIC
0916: 85 76      81            STA   $76          ; SIMULIEREN
0918: 8D 53 0E   82            STA   FILENAME
091B: 20 66 0D   83            JSR   CLEARBUF     ; PUFFER LOESCHEN
                 84  *
                 85  *    MENUE AUSGEBEN
                 86  *
091E: 20 58 FC   87  GETCMD0   JSR   HOME
0921: 20 BD 0D   88            JSR   OUTPUT
0924: A0 A0 A0   89            ASC   "        SOUND - VERWALTUNG 1.0"
0941: 8D 8D      90            HEX   8D8D
0943: A0 A0 A0   91            ASC   "          (C) JOACHIM LANGE"
095D: 8D 8D 8D   92            HEX   8D8D8D
0960: B1 BA A0   93            ASC   "1: EINEN SOUND VOM KEYBOARD HOLEN"
0981: 8D 8D      94            HEX   8D8D
0983: B2 BA A0   95            ASC   "2: SOUND-BANK VOM KEYBOARD HOLEN"
09A3: 8D 8D      96            HEX   8D8D
09A5: B3 BA A0   97            ASC   "3: SPEICHER ZUM KEYBOARD SENDEN"
09C4: 8D 8D      98            HEX   8D8D
09C6: C9 BA A0   99            ASC   "I: SPEICHER-INFO"
09D6: 8D 8D     100            HEX   8D8D
09D8: C3 BA A0  101            ASC   "C: CATALOG"
09E2: 8D 8D     102            HEX   8D8D
09E4: CC BA A0  103            ASC   "L: LOAD SOUND/BANK VON DISK"
09FF: 8D 8D     104            HEX   8D8D
0A01: D3 BA A0  105            ASC   "S: SAVE SPEICHER AUF DISK"
0A1A: 8D 8D     106            HEX   8D8D
0A1C: D1 BA A0  107            ASC   "Q: QUIT                 LAENGE: $"
0A3B: 00        108            HEX   00
                109  *
0A3C: 20 90 0D  110            JSR   CALCLEN      ;DATENLAENGE
0A3F: A5 EB     111            LDA   LENGTH+1     ; BERECHNEN
0A41: A6 EA     112            LDX   LENGTH       ; UND
0A43: 20 41 F9  113            JSR   PRNTAX       ; AUSGEBEN
0A46: 20 8E FD  114            JSR   CROUT
                115  *
                116  *    TASTE ABFRAGEN
                117  *
0A49: 20 BD 0D  118  GETCMD1   JSR   OUTPUT       ;PROMPT
0A4C: 8D        119            HEX   8D           ; AUS-
0A4D: BE        120            ASC   ">" ; GEBEN
0A4E: 00        121            HEX   00
                122  *
                123  *   AUF GUELTIGES KOMMANDO TESTEN
                124  *
0A4F: A9 00     125  GETCMD2   LDA   #$00
0A51: 8D 00 02  126            STA   INBUFF
0A54: 20 0C FD  127            JSR   GETCHAR      ;TAST. LESEN
0A57: C9 B3     128            CMP   #"3"
0A59: F0 2D     129            BEQ   JSRSEND
0A5B: C9 B1     130            CMP   #"1"
0A5D: F0 2F     131            BEQ   JSRGETS
0A5F: C9 B2     132            CMP   #"2"
0A61: F0 31     133            BEQ   JSRGETBK
0A63: C9 D3     134            CMP   #"S"
0A65: F0 39     135            BEQ   JSRSAVE
0A67: C9 CC     136            CMP   #"L"
0A69: F0 2F     137            BEQ   JSRLOAD
0A6B: C9 C3     138            CMP   #"C"
0A6D: F0 3D     139            BEQ   CATALOG
0A6F: C9 C9     140            CMP   #"I"
0A71: F0 33     141            BEQ   JSRINFO
0A73: C9 8D     142            CMP   #$8D         ;CR ?
0A75: F0 32     143            BEQ   JMPGET
0A77: C9 D1     144            CMP   #"Q"
0A79: D0 D4     145            BNE   GETCMD2      ;ERNEUT ABFRAGEN
                146  *
                147  *    PROGRAMMENDE
                148  *
0A7B: 20 58 FC  149            JSR   HOME
0A7E: A9 16     150            LDA   #22
0A80: 85 25     151            STA   VERTCURS
0A82: 20 24 FC  152            JSR   VTABZ
0A85: 4C D3 03  153            JMP   DOSCOLD
                154  *
                155  *  ZUR ENTSPRECHENDEN ROUTINE SPRINGEN
                156  *
0A88: 20 C0 0A  157  JSRSEND   JSR   SENDMEM
0A8B: 4C 1E 09  158            JMP   GETCMD0
0A8E: 20 43 0B  159  JSRGETS   JSR   GETSOUND
0A91: 4C 1E 09  160            JMP   GETCMD0
0A94: 20 63 0B  161  JSRGETBK  JSR   GETBANK
0A97: 4C 1E 09  162            JMP   GETCMD0
0A9A: 20 F3 0B  163  JSRLOAD   JSR   LOAD
0A9D: 4C 1E 09  164            JMP   GETCMD0
0AA0: 20 36 0C  165  JSRSAVE   JSR   SAVE
0AA3: 4C 1E 09  166            JMP   GETCMD0
0AA6: 20 D7 0C  167  JSRINFO   JSR   MEMINFO
0AA9: 4C 1E 09  168  JMPGET    JMP   GETCMD0
                169
                170  ********************************
                171  *
                172  *    CATALOG-KOMMANDO
                173  *
                174  ********************************
                175
0AAC: 20 58 FC  176  CATALOG   JSR   HOME
0AAF: 20 BD 0D  177            JSR   OUTPUT
0AB2: 8D 84     178            HEX   8D84         ;CR/CTRL-D
0AB4: C3 C1 D4  179            ASC   "CATALOG"
0ABB: 8D 00     180            HEX   8D00
0ABD: 4C 49 0A  181            JMP   GETCMD1      ;OHNE MENUE
                182
                183  ********************************
                184  *
                185  *    SPEICHERINHALT SENDEN
                186  *
                187  ********************************
                188
0AC0: 20 9E 0D  189  SENDMEM   JSR   TESTLEN      ;DATENLAENGE
0AC3: D0 03     190            BNE   SENDMEM1     ; <> 0 ?
0AC5: 4C 72 0C  191            JMP   MEMEMPTY     ;MESSAGE AUSGEBEN
0AC8: A9 00     192  SENDMEM1  LDA   #<BUFBEGIN   ;OUTPUT-
0ACA: A0 10     193            LDY   #>BUFBEGIN   ; POINTER
0ACC: 85 FC     194            STA   OUTPTR       ; SETZEN
0ACE: 84 FD     195            STY   OUTPTR+1
0AD0: 20 BD 0D  196            JSR   OUTPUT
0AD3: 88        197            DFB   "H"-$40      ;BACKSPACE
0AD4: D3 D0 C5  198            ASC   "SPEICHER SENDEN:"
0AE4: 00        199            HEX   00
0AE5: 20 09 0E  200            JSR   ACIAINIT
                201  *
                202  *   SPEICHERSCHUTZ AUFHEBEN
                203  *
0AE8: 20 4A 0E  204            JSR   PARMCHG2     ;SYSTEM-EXCLUSIVE
0AEB: 20 DC 0D  205            JSR   MIDIOUT
```

```
0AEE: 77         206          DFB  119         ;MEMORY-PROTECT
0AEF: 00         207          DFB  0           ;OFF !
0AF0: F7         208          DFB  $F7         ;EOX
0AF1: FF         209          DFB  $FF         ;DATENENDE
                 210  *
                 211  *   SPEICHERINHALT SENDEN, BIS EOX
                 212  *   AUFTRITT ODER POINTER $9000
                 213  *   ERREICHT HAT (ERROR)
                 214  *
0AF2: A0 00      215          LDY  #$00
0AF4: A2 90      216          LDX  #$90        ;FUER $9000-TEST
0AF6: AD C0 C0   217  TRANSLOP LDA STATREG     ;ACIA
0AF9: 4A         218          LSR              ;BIT 2
0AFA: 4A         219          LSR              ; TESTEN
0AFB: 90 F9      220          BCC  TRANSLOP
0AFD: B1 FC      221          LDA  (OUTPTR),Y  ;BYTE HOLEN
0AFF: 8D C1 C0   222          STA  TDATREG     ;UND ZUM ACIA
0B02: 8D D0 07   223          STA  $7D0        ;VIDEO-ACTION
0B05: C9 F7      224          CMP  #$F7        ;EOX ?
0B07: F0 23      225          BEQ  READY       ; JA
0B09: E6 FC      226          INC  OUTPTR      ;POINTER=
0B0B: D0 E9      227          BNE  TRANSLOP    ; POINTER+1
0B0D: E6 FD      228          INC  OUTPTR+1
0B0F: E4 FD      229          CPX  OUTPTR+1    ;$9000 ERREICHT ?
0B11: D0 E3      230          BNE  TRANSLOP    ; NEIN
                 231  *
0B13: 20 BD 0D   232          JSR  OUTPUT
0B16: 8D         233          HEX  8D
0B17: D4 D2 C1   234          ASC  "TRANSMIT-ERROR !"
0B27: 8D 00      235          HEX  8D00
0B29: 4C 81 0D   236          JMP  GETKEY      ;ERROR-AUSGANG
                 237  *
                 238  *   SPEICHERSCHUTZ WIEDER EINSCHALTEN
                 239  *
0B2C: 20 4A 0E   240  READY    JSR  PARMCHG2   ;SYSTEM-EXCLUSIVE
0B2F: 20 DC 0D   241          JSR  MIDIOUT
0B32: 77         242          DFB  119         ;MEMORY-PROTECT
0B33: 01         243          DFB  1           ;ON !
0B34: F7         244          DFB  $F7         ;EOX
0B35: FF         245          DFB  $FF         ;DATENENDE
                 246  *
                 247  *   BETRIEBSART 'PLAY ON' EINSTELLEN
                 248  *
0B36: 20 4A 0E   249          JSR  PARMCHG2    ;SYSTEM-EXCLUSIVE
0B39: 20 DC 0D   250          JSR  MIDIOUT
0B3C: 64         251          DFB  100         ;PLAY-MODUS
0B3D: 01         252          DFB  1           ;ON !
0B3E: F7         253          DFB  $F7         ;EOX
0B3F: FF         254          DFB  $FF         ;DATENENDE
                 255  *
0B40: 4C A8 0B   256          JMP  READYMSG    ;SENDEN FERTIG
                 257
                 258  *********************************
                 259  *
                 260  *    EINEN SOUND VOM
                 261  *    SYNTHESIZER ABRUFEN
                 262  *
                 263  *********************************
                 264
0B43: 20 BD 0D   265  GETSOUND JSR OUTPUT
0B46: 88         266          DFB  "H"-$40     ;BACKSPACE
0B47: D3 CF D5   267          ASC  "SOUND HOLEN:"
0B53: 00         268          HEX  00
0B54: 20 66 0D   269          JSR  CLEARBUF
                 270  *
0B57: 20 DC 0D   271  REQUESTS JSR MIDIOUT     ;SOUND ANFORDERN
0B5A: F0         272          DFB  $F0         ;STATUSBYTE
0B5B: 43         273          DFB  $43         ;ID.-NUMMER
0B5C: 20         274          DFB  $20         ;SUBSTATUS REQUEST
0B5D: 03         275          DFB  $03         ;SINGLE SOUND
0B5E: F7         276          DFB  $F7         ;EOX
0B5F: FF         277          DFB  $FF         ;DATENENDE
0B60: 4C 7F 0B   278          JMP  RECEIVE
                 279
                 280  *********************************
                 281  *
                 282  *    GANZE SOUND-BANK VOM
                 283  *    SYNTHESIZER ABRUFEN
                 284  *
                 285  *********************************
                 286
0B63: 20 BD 0D   287  GETBANK  JSR  OUTPUT
0B66: 88         288          DFB  "H"-$40     ;BACKSPACE
0B67: C2 C1 CE   289          ASC  "BANK HOLEN:"
0B72: 00         290          HEX  00
0B73: 20 66 0D   291          JSR  CLEARBUF
                 292  *
0B76: 20 DC 0D   293  REQUESTB JSR MIDIOUT     ;BANK ANFORDERN
0B79: F0         294          DFB  $F0         ;STATUSBYTE
0B7A: 43         295          DFB  $43         ;ID.-NUMMER
0B7B: 20         296          DFB  $20         ;SUBSTATUS REQUEST
0B7C: 04         297          DFB  $04         ;GANZE BANK
0B7D: F7         298          DFB  $F7         ;EOX
0B7E: FF         299          DFB  $FF         ;DATENENDE
                 300  *
                 301  *   BLOCK-RECEIVE BIS DAS BYTE EOX
                 302  *   (=$F7) EINTRIFFT
                 303
0B7F: A2 90      304  RECEIVE  LDX  #$90       ;FUER $9000-TEST
0B81: A0 00      305          LDY  #$00
0B83: AD 00 C0   306  RECEIVE1 LDA  KEYBD      ;TASTE GEDRUECKT ?
0B86: 30 47      307          BMI  RECEND      ; JA
0B88: AD C0 C0   308          LDA  STATREG     ;ACIA
0B8B: 4A         309          LSR              ;BIT 0 TESTEN
0B8C: 90 F5      310          BCC  RECEIVE1    ;NICHTS EMPFANGEN
0B8E: AD C1 C0   311  RECEIVED LDA  RDATREG    ;ACIA
0B91: C9 FE      312          CMP  #$FE        ;ACTIVE SENSING ?
0B93: F0 EE      313          BEQ  RECEIVE1    ;JA,IGNORE
0B95: 91 FA      314          STA  (INPTR),Y
0B97: 8D D0 07   315          STA  $7D0        ;VIDEO-ACTION
0B9A: E6 FA      316          INC  INPTR       ;POINTER=
0B9C: D0 06      317          BNE  EOXTEST     ; POINTER+1
0B9E: E6 FB      318          INC  INPTR+1
0BA0: E4 FB      319          CPX  INPTR+1     ;$9000 ERREICHT ?
0BA2: F0 15      320          BEQ  RECERROR    ; JA
0BA4: C9 F7      321  EOXTEST  CMP  #$F7       ;EOX ?
0BA6: D0 DB      322          BNE  RECEIVE1    ;NEIN
                 323  *
0BA8: 20 BD 0D   324  READYMSG JSR  OUTPUT
0BAB: 8D 8D      325          HEX  8D8D
0BAD: C6 C5 D2   326          ASC  "FERTIG !"
0BB5: 00         327          HEX  00
0BB6: 4C 81 0D   328          JMP  GETKEY      ;RECEIVE BEENDET
                 329  *
                 330  *   ERROR-AUSGANG, BEVOR DAS DOS
                 331  *   UEBERSCHRIEBEN WIRD
                 332  *
0BB9: 20 BD 0D   333  RECERROR JSR  OUTPUT
0BBC: 8D 8D      334          HEX  8D8D
0BBE: D2 C5 C3   335          ASC  "RECEIVE-ERROR !"
0BCD: 8D 00      336          HEX  8D00
                 337  *
0BCF: 20 BD 0D   338  RECEND   JSR  OUTPUT
0BD2: 8D         339          HEX  8D
0BD3: CB C5 C9   340          ASC  "KEINE DATEN EMPFANGEN"
0BE8: 8D 00      341          HEX  8D00
0BEA: 20 66 0D   342          JSR  CLEARBUF
0BED: 8D 10 C0   343          STA  KEYBDSTB
```

```
0BF0: 4C 81 0D  344          JMP  GETKEY      ;ERROR-AUSGANG
                345
                346 ********************************
                347 *
                348 *    LOAD-ROUTINE
                349 *
                350 ********************************
                351
0BF3: 20 BD 0D  352 LOAD     JSR  OUTPUT
0BF6: 88        353          DFB  "H"-$40     ;BACKSPACE
0BF7: CC CF C1  354          ASC  "LOAD : "
0BFE: 00        355          HEX  00
0BFF: 20 16 0E  356          JSR  GETFNAME    ;NAME HOLEN
0C02: B0 31     357          BCS  RETURN
                358 *
0C04: 20 66 0D  359 DOLOAD1  JSR  CLEARBUF
0C07: 20 BD 0D  360          JSR  OUTPUT      ;DOS-BLOAD
0C0A: 8D 84     361          HEX  8D84        ;CR/CTRL-D
0C0C: C2 CC CF  362          ASC  "BLOAD"
0C11: 00        363          HEX  00
0C12: 20 9D 0C  364          JSR  FNAMEOUT    ;NAME AUSGEBEN
0C15: 20 BD 0D  365          JSR  OUTPUT
0C18: AC C1 A4  366          ASC  ",A$1000"
0C1F: 8D 00     367          HEX  8D00
0C21: AD 60 AA  368          LDA  LENGTADR    ;DATENLAENGE
0C24: AC 61 AA  369          LDY  LENGTADR+1  ; EINTRAGEN
0C27: 85 EA     370          STA  LENGTH
0C29: 84 EB     371          STY  LENGTH+1
0C2B: 18        372          CLC
0C2C: 69 00     373          ADC  #<BUFBEGIN ;PUFFER-
0C2E: 85 FA     374          STA  INPTR       ; POINTER
0C30: 98        375          TYA              ; SETZEN
0C31: 69 10     376          ADC  #>BUFBEGIN
0C33: 85 FB     377          STA  INPTR+1
0C35: 60        378 RETURN   RTS
                379
                380 ********************************
                381 *
                382 *    SAVE - ROUTINE
                383 *
                384 ********************************
                385
0C36: 20 9E 0D  386 SAVE     JSR  TESTLEN     ;DATENLAENGE>0 ?
0C39: F0 37     387          BEQ  MEMEMPTY    ; NEIN
0C3B: 20 BD 0D  388          JSR  OUTPUT
0C3E: 88        389          DFB  "H"-$40     ;BACKSPACE
0C3F: D3 C1 D6  390          ASC  "SAVE : "
0C46: 00        391          HEX  00
0C47: 20 16 0E  392          JSR  GETFNAME
0C4A: B0 E9     393          BCS  RETURN      ;KEINE EINGABE
                394 *
0C4C: 20 BD 0D  395 DOSAVE1  JSR  OUTPUT      ;DOS-BSAVE
0C4F: 8D 84     396          HEX  8D84        ;CR/CTRL-D
0C51: C2 D3 C1  397          ASC  "BSAVE"
0C56: 00        398          HEX  00
0C57: 20 9D 0C  399          JSR  FNAMEOUT    ;NAME AUSGEBEN
0C5A: 20 BD 0D  400 DOSAVE3  JSR  OUTPUT
0C5D: AC C1 A4  401          ASC  ",A$1000,L$"
0C67: 00        402          HEX  00
0C68: A5 EB     403          LDA  LENGTH+1    ;DATENLAENGE
0C6A: A6 EA     404          LDX  LENGTH      ; ANGEBEN
0C6C: 20 41 F9  405          JSR  PRNTAX
0C6F: 4C 8E FD  406          JMP  CROUT       ;SAVE-ENDE
                407 *
                408 *   'SPEICHER LEER' AUSGEBEN
                409 *
0C72: 20 BD 0D  410 MEMEMPTY JSR  OUTPUT
0C75: 88        411          DFB  "H"-$40
0C76: D3 D0 C5  412          ASC  "SPEICHER LEER !"
0C85: 8D 00     413          HEX  8D00
0C87: 4C 81 0D  414          JMP  GETKEY      ;EINGABE ABWARTEN
                415 *
                416 *    FILENAME AUSGEBEN
                417 *
0C8A: 20 BD 0D  418 FNAMOUT0 JSR  OUTPUT
0C8D: 8D 8D     419          HEX  8D8D
0C8F: A0 A0 A0  420          ASC  "    FILENAME :"
0C9C: 00        421          HEX  00
0C9D: A0 00     422 FNAMEOUT LDY  #$00
0C9F: B9 53 0E  423 FNAMLOOP LDA  FILENAME,Y
0CA2: F0 06     424          BEQ  FNOUTRET    ;0=STRING-ENDE
0CA4: 20 ED FD  425          JSR  COUT
0CA7: C8        426          INY
0CA8: D0 F5     427          BNE  FNAMLOOP    ;NEXT CHAR.
0CAA: 60        428 FNOUTRET RTS
                429
                430 ********************************
                431 *
                432 *   SPEICHER-INFO: ALLE SOUNDNAMEN
                433 *   UND GGF. FILENAME AUSGEBEN
                434 *
                435 * EINZELSOUND :
                436 *    BLOCKLAENGE    : $65
                437 *    NAMENSPOSITION: $53
                438 *
                439 * SOUND-BANK:
                440 *    BLOCKLAENGE    : 32*$80+8=$1008
                441 *    NAMENSPOSITION: N*$80+$3F
                442 *
                443 ********************************
                444 *
                445 *   HIER SOUNDINFO AUSGEBEN:
                446 *
0CAB: 20 58 FC  447 SOUNDINF JSR  HOME        ;IN DIE MITTE
0CAE: A9 0C     448          LDA  #12         ; DES
0CB0: 85 25     449          STA  VERTCURS    ; BILDSCHIRMS
0CB2: 20 24 FC  450          JSR  VTABZ
0CB5: 20 BD 0D  451          JSR  OUTPUT
0CB8: A0 D3 CF  452          ASC  " SOUND NAME :"
0CC5: 00        453          HEX  00
                454 *
                455 *    SOUND-NAME AUS SPEICHER HOLEN
                456 *
0CC6: A0 53     457          LDY  #$53        ;OFFSET F. NAME
0CC8: B9 00 10  458 SINFLOOP LDA  $1000,Y
0CCB: 09 80     459          ORA  #%10000000 ;SETZE BIT 7
0CCD: 20 F0 FD  460          JSR  COUT1
0CD0: C8        461          INY
0CD1: C0 5D     462          CPY  #$53+10     ;10 CHARACTERS
0CD3: D0 F3     463          BNE  SINFLOOP
0CD5: F0 49     464          BEQ  SHOWFNAM    ;FILENAME
                465 *
                466 *   TEST: SINGLE SOUND ODER BANK
                467 *
0CD7: 20 9E 0D  468 MEMINFO  JSR  TESTLEN     ;DATENLAENGE
0CDA: F0 96     469          BEQ  MEMEMPTY
0CDC: A5 EB     470          LDA  LENGTH+1    ;SINGLE?
0CDE: F0 CB     471          BEQ  SOUNDINF    ;JA
                472 *
                473 *   HIER BANK-INFO AUSGEBEN
                474 *
0CE0: 20 58 FC  475 BANKINFO JSR  HOME
0CE3: 20 BD 0D  476          JSR  OUTPUT
0CE6: A0 A0 A0  477          ASC  "         BANK - INFO"
0CFB: 8D 8D 8D  478          HEX  8D8D8D00
0CFF: A9 00     479          LDA  #0
0D01: 85 E0     480          STA  LINECNT     ;ZEILENZAEHLER
0D03: A0 3F     481          LDY  #<$1000+$3F ;POINTER-
```

```
0D05: A9 10     482          LDA  #>$1000+$3F ; BASISADRESSE
0D07: 84 EC     483          STY  PTR
0D09: 85 ED     484          STA  PTR+1
0D0B: 18        485          CLC
0D0C: 69 08     486          ADC  #$08        ;2. POINTER UM
0D0E: 85 EF     487          STA  PTR2+1      ; $800 VERSETZT
0D10: 84 EE     488          STY  PTR2
0D12: 20 26 0D  489 NAMELOOP JSR  NAMEOUT     ;2 NAMEN/ZEILE
0D15: 20 4F 0D  490          JSR  INC128
0D18: E6 E0     491          INC  LINECNT     ;ZEILE=
0D1A: A5 E0     492          LDA  LINECNT     ; ZEILE+1
0D1C: C9 10     493          CMP  #16         ;16 ZEILEN ?
0D1E: D0 F2     494          BNE  NAMELOOP    ; NEIN
0D20: 20 8A 0C  495 SHOWFNAM JSR  FNAMOUT0
0D23: 4C 7E 0D  496          JMP  GETKEY0     ;ENDE INFO
                497 *
                498 * 2 NAMEN IN EINER ZEILE AUSGEBEN
                499 *
0D26: A2 04     500 NAMEOUT  LDX  #4          ;ERST 4 BLANKS
0D28: 20 4A F9  501          JSR  PRBL2       ; AUSGEBEN
0D2B: A0 00     502          LDY  #$00
0D2D: B1 EC     503 NAMLOOP1 LDA  (PTR),Y     ;1. NAME
0D2F: 09 80     504          ORA  #%10000000  ;SETZE BIT 7
0D31: 20 F0 FD  505          JSR  COUT1
0D34: C8        506          INY
0D35: C0 0A     507          CPY  #10         ;10 CHARACTERS
0D37: D0 F4     508          BNE  NAMLOOP1
0D39: A2 04     509          LDX  #4          ;NOCH 4 BLANKS
0D3B: 20 4A F9  510          JSR  PRBL2       ; AUSGEBEN
0D3E: A0 00     511          LDY  #$00
0D40: B1 EE     512 NAMLOOP3 LDA  (PTR2),Y    ;2. NAME
0D42: 09 80     513          ORA  #%10000000  ;SETZE BIT 7
0D44: 20 F0 FD  514          JSR  COUT1
0D47: C8        515          INY
0D48: C0 0A     516          CPY  #10         ;10 CHARACTERS
0D4A: D0 F4     517          BNE  NAMLOOP3
0D4C: 4C 8E FD  518          JMP  CROUT       ;RETURN
                519 *
                520 * BEIDE POINTER UM 128 BYTES ERHOEHEN
                521 *
0D4F: A5 EC     522 INC128   LDA  PTR         ;1. POINTER
0D51: 18        523          CLC
0D52: 69 80     524          ADC  #128
0D54: 90 02     525          BCC  STOREPTR
0D56: E6 ED     526          INC  PTR+1
0D58: 85 EC     527 STOREPTR STA  PTR
0D5A: A5 EE     528 INC2     LDA  PTR2        ;2. POINTER
0D5C: 18        529          CLC
0D5D: 69 80     530          ADC  #128
0D5F: 90 02     531          BCC  INCRET
0D61: E6 EF     532          INC  PTR2+1
0D63: 85 EE     533 INCRET   STA  PTR2
0D65: 60        534          RTS
                535
                536 **********************************
                537 *
                538 *   DIVERSE UNTERPROGRAMME
                539 *
                540 **********************************
                541 *
                542 *   CLEAR DATENPUFFER
                543 *
0D66: 20 A8 0D  544 CLEARBUF JSR  BEGINBUF    ;POINTER SETZEN
0D69: A9 00     545          LDA  #$00
0D6B: A2 12     546          LDX  #18         ;ANZAHL PAGES
0D6D: A8        547          TAY
0D6E: 91 FA     548 CLRLOOP  STA  (INPTR),Y   ;Y=0
0D70: C8        549          INY
0D71: D0 FB     550          BNE  CLRLOOP
0D73: E6 FB     551          INC  INPTR+1
0D75: CA        552          DEX
0D76: D0 F6     553          BNE  CLRLOOP
0D78: 20 09 0E  554          JSR  ACIAINIT
0D7B: 4C A8 0D  555          JMP  BEGINBUF    ;POINTER SETZEN
                556 *
                557 *   TASTENDRUCK ABWARTEN
                558 *
0D7E: 20 8E FD  559 GETKEY0  JSR  CROUT
0D81: 20 BD 0D  560 GETKEY   JSR  OUTPUT      ;MESSAGE
0D84: 8D        561          HEX  8D
0D85: D4 C1 D3  562          ASC  "TASTE >"
0D8C: 00        563          HEX  00
0D8D: 4C 0C FD  564          JMP  GETCHAR
                565 *
                566 *   PUFFERLAENGE BERECHNEN
                567 *
0D90: A5 FA     568 CALCLEN  LDA  INPTR       ;ANFANGSADRESSE
0D92: 38        569          SEC              ; VON
0D93: E9 00     570          SBC  #<BUFBEGIN  ; ENDADRESSE
0D95: 85 EA     571          STA  LENGTH      ; SUBTRAHIEREN
0D97: A5 FB     572          LDA  INPTR+1
0D99: E9 10     573          SBC  #>BUFBEGIN
0D9B: 85 EB     574          STA  LENGTH+1
0D9D: 60        575          RTS
                576 *
                577 *   TEST AUF DATENLAENGE <> 0
                578 *
0D9E: 20 90 0D  579 TESTLEN  JSR  CALCLEN     ;ERST BERECHNEN
0DA1: A5 EA     580          LDA  LENGTH
0DA3: D0 02     581          BNE  TESTRET     ;SCHON > 0 !
0DA5: A5 EB     582          LDA  LENGTH+1
0DA7: 60        583 TESTRET  RTS
                584 *
                585 *  INPTR AUF PUFFERANFANG SETZEN
                586 *
0DA8: A9 00     587 BEGINBUF LDA  #<BUFBEGIN
0DAA: A0 10     588          LDY  #>BUFBEGIN
0DAC: 85 FA     589          STA  INPTR
0DAE: 84 FB     590          STY  INPTR+1
0DB0: 60        591          RTS
                592 *
                593 *   DOS - ERROR ABFANGEN
                594 *
0DB1: A2 FF     595 ERRSUB   LDX  #$FF        ;STACKPOINTER
0DB3: 9A        596          TXS              ; NEU
0DB4: A9 09     597          LDA  #>GETCMD0-1 ; SETZEN
0DB6: 48        598          PHA
0DB7: A9 1D     599          LDA  #<GETCMD0-1
0DB9: 48        600          PHA
0DBA: 4C 81 0D  601          JMP  GETKEY      ;RTS=JMP GETCMD0
                602 *
                603 *   TEXT-OUTPUT-ROUTINE
                604 *
0DBD: 68        605 OUTPUT   PLA              ;RETTE PC
0DBE: 85 E2     606          STA  TEMP
0DC0: 68        607          PLA
0DC1: 85 E3     608          STA  TEMP+1
0DC3: A0 00     609          LDY  #$00
0DC5: E6 E2     610 OUTLOOP  INC  TEMP
0DC7: D0 02     611          BNE  LOADBYTE
0DC9: E6 E3     612          INC  TEMP+1
0DCB: B1 E2     613 LOADBYTE LDA  (TEMP),Y
0DCD: F0 06     614          BEQ  OUTRET      ;TEXT-DATENENDE ?
0DCF: 20 ED FD  615          JSR  COUT
0DD2: 4C C5 0D  616          JMP  OUTLOOP
0DD5: A5 E3     617 OUTRET   LDA  TEMP+1      ;RESTORE PC
0DD7: 48        618          PHA
0DD8: A5 E2     619          LDA  TEMP
```

```
0DDA: 48          620            PHA
0DDB: 60          621            RTS
                  622  *
                  623  *   MIDI-OUTPUT-ROUTINE: AUSGABE
                  624  *   UEBER ACIA, BIS $FF AUFTRITT
                  625  *
0DDC: 68          626  MIDIOUT   PLA                 ;RETTE PC
0DDD: 85 E2       627            STA   TEMP
0DDF: 68          628            PLA
0DE0: 85 E3       629            STA   TEMP+1
0DE2: A0 00       630            LDY   #$00
0DE4: E6 E2       631  MOUTLOOP  INC   TEMP
0DE6: D0 02       632            BNE   GETBYTE
0DE8: E6 E3       633            INC   TEMP+1
0DEA: B1 E2       634  GETBYTE   LDA   (TEMP),Y
0DEC: C9 FF       635            CMP   #$FF          ;MIDI-DATENENDE ?
0DEE: F0 06       636            BEQ   MDOUTRET      ; JA
0DF0: 20 FD 0D    637            JSR   ACIAOUT
0DF3: 4C E4 0D    638            JMP   MOUTLOOP
0DF6: A5 E3       639  MDOUTRET  LDA   TEMP+1        ;RESTORE PC
0DF8: 48          640            PHA
0DF9: A5 E2       641            LDA   TEMP
0DFB: 48          642            PHA
0DFC: 60          643            RTS
                  644  *
                  645  *   ACIA-OUT-ROUTINE
                  646  *
0DFD: AA          647  ACIAOUT   TAX                 ;RETTE AKKU
0DFE: A9 02       648            LDA   #%00000010 ;TD-REGISTER
0E00: 2C C0 C0    649  OUTWAIT   BIT   STATREG       ; LEER?
0E03: F0 FB       650            BEQ   OUTWAIT       ; NEIN!
0E05: 8E C1 C0    651            STX   TDATREG       ;AKKUINHALT
0E08: 60          652            RTS
                  653  *
                  654  *   ACIA-INITIALISIERUNG
                  655  *   8 BIT, 1 STOP, TEILERRATE 64
                  656  *
0E09: 20 10 0E    657  ACIAINIT  JSR   RESETACI      ;ERST RESET
0E0C: A9 16       658            LDA   #%00010110 ; DANN
0E0E: D0 02       659            BNE   SETREG        ; SETZEN
0E10: A9 03       660  RESETACI  LDA   #%00000011
0E12: 8D C0 C0    661  SETREG    STA   CTRLREG       ;ACIA
0E15: 60          662            RTS
                  663  *
                  664  *   DATEINAME EINGEBEN
                  665  *
0E16: A9 00       666  GETFNAME  LDA   #$00
0E18: 8D 53 0E    667            STA   FILENAME
0E1B: A9 88       668  JSRINPUT  LDA   #"H"-$40      ;BACKSPACE
0E1D: 85 33       669            STA   PROMPT
0E1F: 20 6A FD    670            JSR   GETLN
0E22: A9 00       671            LDA   #$00          ;ENDE MARKIEREN
0E24: 9D 00 02    672            STA   INBUFF,X
0E27: AD 00 02    673            LDA   INBUFF        ;EINGABE GEMACHT ?
0E2A: F0 13       674            BEQ   NOINPUT       ; NEIN
                  675  *
0E2C: A0 00       676            LDY   #$00          ;FILENAMEN
0E2E: B9 00 02    677  COPYLOOP  LDA   INBUFF,Y      ; KOPIEREN
0E31: 99 53 0E    678            STA   FILENAME,Y
0E34: C8          679            INY
0E35: C0 1E       680            CPY   #30           ;BYTE 0...29
0E37: F0 04       681            BEQ   FCOPYEND
0E39: C9 00       682            CMP   #$00          ;0=ENDE FILENAME
0E3B: D0 F1       683            BNE   COPYLOOP
0E3D: 18          684  FCOPYEND  CLC                 ;C=0->OK
0E3E: 60          685            RTS
0E3F: 38          686  NOINPUT   SEC                 ;C=1->KEINE EINGABE
0E40: 60          687            RTS
                  688  *
                  689  *   SOFTWARE-TASTENBETAETIGUNG
                  690  *
0E41: 20 DC 0D    691  PARMCHG1  JSR   MIDIOUT
0E44: F0          692            DFB   $F0           ;STATUSBYTE
0E45: 43          693            DFB   $43           ;ID.-NUMMER
0E46: 10          694            DFB   $10           ;SUBSTATUS
0E47: 08          695            DFB   $08           ;PARAMETER-ART
0E48: FF          696            DFB   $FF           ;DATENENDE
0E49: 60          697            RTS
                  698  *
                  699  *   DIREKTE PARAMETERAENDERUNG
                  700  *  (OHNE FUNKTIONSTASTENSIMULATION)
                  701  *
0E4A: 20 DC 0D    702  PARMCHG2  JSR   MIDIOUT
0E4D: F0          703            DFB   $F0           ;STATUSBYTE
0E4E: 43          704            DFB   $43           ;ID.-NUMMER
0E4F: 10          705            DFB   $10           ;SUBSTATUS
0E50: 12          706            DFB   $12           ;PARAMETER-ART
0E51: FF          707            DFB   $FF           ;DATENENDE
0E52: 60          708            RTS
                  709  *
                  710  FILENAME

--End assembly, 1363 bytes, Errors: 0
```

wird immer der gerade aktive (im Arbeitsspeicher befindliche) Sound übertragen.

2: Sound-Bank vom Keyboard holen

Analog zu „1“, nur daß die ganze Internal-Sound-Bank zur Ausgabe angefordert wird. Der vom DX-27 gesendete Datenblock ist einschließlich Vorspann $1008 Bytes lang, wenn er richtig empfangen wurde (4104 Bytes, zusammengesetzt aus 6 Vorspann-, 4096 Nettodaten-, 1 Prüfsumme-, 1 EOX-Bytes). Er enthält 32 128-Byte-Blöcke, in denen die Klangparameter der einzelnen Sounds in komprimierter Form vorliegen. Da die Klangparameter-Daten nicht die volle Blockgröße von 128 Bytes nutzen, sind die Einzelblöcke am Ende mit Nullen aufgefüllt. Von den 32 Einzel-Sounds sind die letzten acht nicht belegt, da im DX-27 nur 24 Speicherplätze zur Verfügung stehen (Datenformat kompatibel zum DX-21, der 32 Speicherplätze besitzt).

3: Speicher zum Keyboard senden

Der Speicherinhalt ab Adresse $1000 wird über die Schnittstelle ausgegeben. Das Programm überträgt so lange, bis es ein EOX ($F7) im Speicher findet. Mit diesem Befehl lassen sich sowohl einzelne Sounds als auch ganze Bänke zum Synthesizer schicken (siehe Datenformat). Vor dem Senden prüft das Programm, ob überhaupt Daten im Speicher stehen und gibt gegebenenfalls die Meldung „Speicher leer“ aus. Damit der Synthesizer die Daten auch wirklich aufnimmt, wird vor der Übertragung des Datenblocks der Speicherschreibschutz seines CMOS-RAMs softwaremäßig aufgehoben und nach Übertragungsende wieder aktiviert. Das birgt natürlich ein gewisses Risiko in sich, da man eventuell wertvolle Sounds mit ungültigen Daten überschreibt, also Vorsicht! Auch hier ist die Notbremse eingebaut, damit das Programm im Fehlerfall (wenn es kein EOX im Speicher findet) nicht unendlich lang Daten überträgt. Obwohl das Programm sich nicht aufhängt, muß man in diesem Fall damit rechnen, daß dem Synthesizer unbrauchbare Daten übermittelt worden sind (Fehlermeldung „TRANSMIT-ERROR“). Wie bereits beschrieben, erkennt der Synthesizer automatisch am Vorspann die Art der Daten, daher ist die Fallunterscheidung Sound/Bank nicht nötig.

I: Speicher-Info

Dient zur Information, welche Sounds gerade im Speicher stehen. Abhängig

von den im Speicher befindlichen Daten wird der Name des Einzel-Sounds oder eine Tabelle von 32 Sound-Namen einer Sound-Bank ausgegeben. Außerdem wird der zuletzt benutzte Dateiname angezeigt. Die Daten können entweder von Diskette geladen oder vom Synthesizer empfangen worden sein. Bei leerem Speicher wird die entsprechende Meldung ausgegeben.

C: Catalog

Verkürzter Catalog-Befehl. Er erleichtert vor allem das Suchen und Eingeben (mit den üblichen Escape-Sequenzen) von Dateinamen beim Laden und Sichern von Daten. Nachdem das Inhaltsverzeichnis ausgegeben ist, erscheint nicht das Menü, sondern nur das Prompt-Symbol „>", bis der nächste Befehl abgearbeitet oder Return eingegeben wurde.

L: Load Sound/Bank von Disk

Ein Binärfile wird von Diskette in den Speicher geladen. Bei der Angabe des Dateinamens kann man Slot und Drive nach den DOS-Konventionen angeben.

Es sollten natürlich sinnvolle Daten, also ein Einzel-Sound oder eine Sound-Bank, geladen werden. Andernfalls würde man dem Synthesizer unsinnige Daten übermitteln.

S: Save Speicher auf Disk

Der vom Synthesizer empfangene Datenblock (Einzel-Sound oder Sound-Bank) wird unter dem zugewiesenen Dateinamen auf Diskette gesichert. Das Programm überprüft, ob überhaupt Daten im Speicher stehen und „verweigert" das Sichern eines leeren Speichers. Ansonsten gilt das unter „LOAD" Gesagte.

Q: Quit

Beenden des Programms. Ein erneuter Start ist mit „& Return" möglich, solange man das Programm nicht zerstört hat und der Ampersand-Vektor nicht verändert wird.

Bei allen hier beschriebenen Kommunikationsvorgängen muß natürlich die MIDI-Schnittstelle des Synthesizers aktiviert sein, und der Einstellparameter SYS-INFO muß auf „On" stehen, damit sich der Synthesizer bei SYSTEM-EXCLUSIVE-Informationen nicht taub stellt. Um so flexibel wie möglich zu bleiben, benutzt das Programm immer das EOX-Byte zum Erkennen des Datenendes, es können daher Daten von beliebiger Länge (sofern sie in den Apple-Speicher passen) verarbeitet werden. Als DX-27-spezifisch ist hier noch das Auslesen der Sound-Namen zu nennen, diese Daten stehen „irgendwo" im Sound-Datenblock.

Mehr Komfort ist denkbar

Wem das getrennte Abrufen und Sichern von Daten trotz der wenigen Tastenbetätigungen noch zu umständlich ist, dem fallen sicher ein paar Assemblerzeilen ein, um beide Vorgänge zu einem zusammenzufassen.

Das hier vorgestellte Programm stellt das Minimum dar, um Sound-Daten zu verwalten. Es dient hauptsächlich zum Übertragen von Sounds von einem Speichermedium auf ein anderes. Hat man die Daten erst einmal auf Diskette, kann man sie nach Belieben weiterbearbeiten. Mit ein paar PEEK- und POKE-Befehlen in Basic lassen sich zum Beispiel die Sound-Blöcke in ihrer Reihenfolge umgruppieren oder Sounds aus verschiedenen Dateien zusammenkopieren. Oder man gibt seinen Sound-Kreationen neue Namen. Solche Funktionen sind, direkt am Synthesizer vollzogen, relativ umständlich zu handhaben, mit einem kleinen Programm lassen sie sich sehr vereinfachen.

Vom Autor kann eine fertig angepaßte Version des Programms für den ebenfalls weit verbreiteten Synthesizer Yamaha DX-7 bezogen werden (Diskette und Bedienungsanleitung).

Literatur

[1] *Lange, Joachim:* MIDI-Interface für den Apple-II. mc 1986, Heft 12, Seite 60...68.
[2] DX-27, Digital programmierbarer Algorithmus-Synthesizer, Bedienungsanleitung. Yamaha, Oktober 1985.
[3] *Philipp, Siegfried:* MIDI-Kompendium 2. Verlag Kapehl und Philipp, Fränkisch Krumbach 1986.

Titan, eine PC-Volltext-Datenbank

PublicSoft, eine Tochter von Bertelsmann, die sich auf die Vermarktung preiswerter Mikrocomputersoftware für jedermann spezialisiert hat, bietet ab sofort eine Volltext-Datenbank für den PC an. Zu einem Preis von 1099.– DM erhält man ein Programmpaket auf zwei Disketten, das etwa ½ MByte Umfang hat. Dieses Datenbankprogramm ist komplett menügesteuert und auch für den PC-Laien nach anfänglichem Training leicht bedienbar. Es besitzt Schnittstellen zu Wordstar (3.2 und 2000), Tex-Ass, MS-Word (alle Versionen) und für reine ASCII-Dateien. In der Redaktion konnten im Test reine ASCII-Dateien in die Datenbank übernommen werden. Dabei leistete das System Schwerstarbeit, denn es mußte während des Überspielens die Indexierung durchführen, was bedeutet, daß der Text aufgelöst in eine Indexwörter-Datei mit entsprechenden Trefferzeigern und im Original abgespeichert werden mußte. Der Lohn für diese Mühe, die auf einem XT (das Programm ist nur sinnvoll, wenn eine Festplatte zur Verfügung steht) für etwa 1 KByte Text drei Minuten dauerte, war dann die Fähigkeit des Systems, in einer Recherche blitzschnell alle gewünschten Suchwörter im Text aufzufinden und sowohl die Fundstelle als auch deren Umgebung auf dem Bildschirm darzustellen. Natürlich läßt sich das Ergebnis einer Suche auch drucken. Die Kapazitätsgrenze des Systems beträgt von der internen Logik her 190 000 Zeilen zu je 80 Zeichen. Dann wird allerdings für alle noch notwendigen Hilfsdateien und Organisations- sowie Reorganisationsläufe eine Festplatte mit 32 KByte benötigt. Das System bietet eine vom Benutzer beeinflußbare Stop-Wortliste und auch einen Maskengenerator. Damit erzeugte Datensätze werden wie Volltext gespeichert und durchsucht. Diese „horizontale" Software ist für Leute, die viel Literatur-Recherche betreiben müssen, bestens geeignet. Abstract für Abstract kann ein Forscher an der Uni zum Beispiel eine sehr umfangreiche Spezialbibliographie aufbauen. Oder es können in einem Büro alle Gesprächsprotokolle, Verträge und anderes abgelegt werden und nach Stichwort schnell und präzise gefunden werden. Nach Angaben der Firma PublicSoft soll bald auch eine netzwerkfähige Version auf den Markt kommen, so daß das System dann nicht nur an einem Arbeitsplatz nutzbar ist.

Ro nach Unterlagen von PublicSoft.

Klaus Bovermann

Das Betriebssystem Eumel-Elan

Jetzt für den Atari ST verfügbar

Der Reigen der Programmiersprachen, die auf dem Atari ST implementiert sind, ist bereits sehr bunt. Neben Basic, Pascal, verschiedenen C-Compilern, Fortran und Modula werden auch Sprachen der „künstlichen Intelligenz" wie Lisp und Prolog angeboten. Jetzt hat sich Eumel-Elan dazu gesellt.

Kauft man sich eine Programmiersprache für den Atari ST, so erhält man meist eine Sammlung vieler verschiedener Dateien. In der Regel gibt es einen (mehr oder weniger brauchbaren) Editor, daneben Interpreter und/oder Compiler-, Linker-, Debugger-Programme sowie massenweise Hilfsdateien. Hat man ein besonders gutes System erstanden, sind viele dieser Teile unter einem gemeinsamen DESK-Programm zusammengefaßt und aufrufbar. Teile vom Atari-TOS sind eingebunden, hauptsächlich die Dateioperationen. Somit hat man es, so meint man, mit einem Betriebssystem zu tun, in dem eine bestimmte Programmiersprache eingebunden ist.

```
PROC löse quadratische gleichung (REAL CONST p,q):
  IF   diskriminante negativ
       THEN putline (keine lösung)
  ELIF diskriminante null
       THEN putline (eine lösung)
       ELSE putline (zwei lösungen)
  FI.

  diskriminante negativ:
    diskriminante < 0.0.

  diskriminante null:
    diskriminante = 0.0.

  diskriminante:
    p*p/4.0 - q.

  keine lösung:
    "Die Gleichung hat keine Lösung".

  eine lösung:
    "Die Gleichung hat als einzige Lösung
    den Wert x = " + text (einziger wert).

  zwei lösungen:
    "Lösungen sind: x1 = " + text (erste)
    + " x2 = " + text (zweite).

  einziger wert:
    -p/2.0.

  erste:
    einziger wert + sqrt (diskriminante).

  zweite:
    einziger wert - sqrt (diskriminante)
END PROC löse quadratische gleichung;

putline ("Lösung eine quadratischen Glei-");
putline ("chung der Form x^2 + px + q = 0");
REAL VAR p,q;
put ("p ="); get (p);
put ("q ="); get (q);
löse quadratische gleichung (p,q)
```

Bild 1. In Elan wird „Top-Down" programmiert

Doch weit gefehlt. Wo bleibt die Druckeransteuerung, um Programmlistings ausgeben zu können? Wer kümmert sich um den Datentransport von der Festplatte zum Floppylaufwerk? Weshalb kann man vorhandenen Speicherplatz nicht komplett nutzen? Hat man eine Festplatte, so dient diese hauptsächlich als Massenspeicher. Während der Arbeit sind maximal 1 MByte RAM im Hauptspeicher ansprechbar. Der massenhafte Platz auf der Platte könnte vom Betriebssystem genutzt und verwaltet werden. Darüber hinaus hat man mit der Motorola-CPU 68000 einen leistungsstarken Prozessor, der von den vorhandenen Systemen in der Regel nur minimal ausgelastet wird. Ist beispielsweise der Editor aufgerufen (dies ist sicherlich die am häufigsten ausgeübte Tätigkeit des Atari ST-Benutzers), so hat der Prozessor kaum Arbeit. In dieser Zeit könnten auch andere Aufgaben bearbeitet werden, wie z. B. die Druckausgabe, das Sichern von Dateien zur Diskette oder ähnliche Dienstleistungen.

Hier nun setzt das EUMEL-Betriebssystem an. EUMEL (Extendable multi-User Microprozessor Elan) wurde von der Universität Bielefeld und der GMD (Gesellschaft für Mathematik und Datenverarbeitung) in Bonn entwickelt und bereits auf unterschiedlichen Mikroprozessoren (Z80, 8086, 80186, 80286, 80386, Z8001) implementiert. Seit kurzer Zeit läuft dieses Betriebssystem auch mit der CPU 68000 des Atari ST. Dieses Betriebssystem enthält neben verschiedenen Dienstleistungsprogrammen, weiteren Compilern und Interpretern einen Elan-Compiler, der im folgenden beschrieben wird.

Die Sprache Elan

Elan (Elaborated Language) ist eine Sprache der Algol-Familie und erlaubt in vorbildlicher Weise das Codieren von Algorithmen in konsequenter „TOP-DOWN-Strategie". Darüber hinaus lebt die Sprache von Modularisierungen, die

```
LET anzahl = 5;

PROC lese die daten ein (ROW anzahl TEXT VAR feld):
  INT VAR nummer;
  FOR nummer FROM 1 UPTO anzahl REP
    put ("Inhalt des " + text (nummer) + ". Feld ");
    getline (feld [nummer])
  END REPEAT
END PROC lese die daten ein;

PROC sortiere (ROW anzahl TEXT VAR feld):
  INT VAR i,j;
  FOR i FROM 1 UPTO anzahl - 1 REP
    FOR j FROM i + 1 UPTO anzahl REP
      IF feld [i] > feld [j]
         THEN tausche (feld [i], feld [j])
      FI
    END REP
  END REP
END PROC sortiere;

PROC tausche (TEXT VAR erster, zweiter):
  TEXT VAR dummy :: erster;
  erster := zweiter;
  zweiter := dummy
END PROC tausche;

PROC ausgabe (ROW anzahl TEXT CONST feld):
  INT VAR nummer;
  FOR nummer FROM 1 UPTO anzahl REP
    put ("Inhalt des " + text (nummer) + ". Feld ");
    putline (feld [nummer])
  END REPEAT
END PROC ausgabe;

ROW anzahl TEXT VAR namen;

lese die daten ein (namen);
sortiere (namen);
ausgabe (namen)
```

Bild 2. Felder können als Zusammenfassung gleichartiger Objekte erzeugt werden

in dieser Ausführlichkeit von keiner anderen Sprache (evtl. in Ansätzen von Modula) erreicht wird. Somit eignet sich das Sprachkonzept von Elan in hervorragender Weise zum Einsatz in Schule und Ausbildung.
Ein Programm in Elan könnte etwa so aussehen wie in *Bild 1* gezeigt.
Das Beispiel zeigt die konsequente Verwendung der TOP-DOWN-Strategie, die in Elan in der Regel zweistufig abläuft. Ein Elan-Programm setzt sich dabei aus einzelnen Prozeduren zusammen, die wiederum in sog. Refinements unterteilt sind. Refinements sind dabei parameterlose Anweisungsfolgen, die Zugriff auf alle Daten der übergeordneten Prozedur haben. Dabei kennen Refinements selber keine lokalen Daten. Wesentlich ist, daß solche Refinements auch Objekte liefern können, wie z. B. *diskriminante:*
$p \times p/4.0-q$.
Prozeduren sind algorithmische Abstraktionen der 2. Stufe. Sie können verschiedene Parameter besitzen, haben in der Regel einen eigenen, lokalen Variablenraum und können ebenfalls Objekte liefern.
Elementare Datentypen sind in Elan die Texte (TEXT), ganze Zahlen (INT), reelle Zahlen (REAL) und Wahrheitswerte (BOOL). Zum wissenschaftlichen Bereich gehören die Datentypen LONGINT (ganze Zahlen, die dynamisch bis zu 32 767 Ziffern umfassen können), VECTOR und MATRIX. Daneben können Felder unterschiedlicher Größe und Dimension als Zusammenfassung gleichartiger Objekte sowie Strukturen von verschiedenen Datentypen erzeugt werden.

```
TYPE MENSCH = STRUCT (TEXT name, INT vermögen, BOOL nett);

LET anzahl = 10;

TYPE VERWANDTE = ROW anzahl MENSCH;

PROC zeige alle netten verwandten (VERWANDTE CONST meine familie):
  INT VAR nummer;
  FOR nummer FROM 1 UPTO anzahl REP
    IF der mensch ist nett
       THEN putline (person + " ist sehr nett.")
    FI
  PER.

  der mensch ist nett:
    meine familie [nummer].nett.

  person:
    meine familie [nummer].name
END PROC zeige alle netten verwandten;

PROC pruefe alle verwandten (VERWANDTE VAR meine familie):
  INT VAR nummer;
  FOR nummer FROM 1 UPTO anzahl REP
    IF der mensch hat genug geld
       THEN er ist nett
       ELSE er ist mir unsympatisch
    FI
  PER.

  der mensch hat genug geld:
    der mensch.vermögen > 10000.

  er ist nett:
    der mensch.nett := TRUE.

  der mensch:
    meine familie [nummer].

  er ist mir unsympatisch:
    der mensch.nett := FALSE
END PROC pruefe alle verwandten;

PROC erfrage verwandte (VERWANDTE VAR meine familie):
  INT VAR nummer;
  FOR nummer FROM 1 UPTO anzahl REP
    put ("Wie heißt der Verwandte :");
    getline (meine familie [nummer].name);
    put ("Wie hoch ist sein Gehalt:");
    get (meine familie [nummer].vermögen)
  PER END PROC erfrage verwandte;.

VERWANDTE VAR die familie;

erfrage verwandte (die familie);
pruefe alle verwandten (die familie);
zeige alle netten verwandten (die familie)
```

Bild 4. Eigene Datentypen werden definiert

Ein entsprechendes Beispiel zeigt *Bild 2*. In Elan ist es möglich, Prozeduren als Parameter an andere Prozeduren zu übergeben (*Bild 3*).
Wesentliche Eigenschaft von Elan ist die Möglichkeit, eigene Datentypen zu definieren. Auch dazu ein Beispiel (*Bild 4*). Eine weitere Modularisierungsebene besteht in der Strukturierung durch sogenannte Pakete. Man benutzt sie, um folgenden Anwendungen neue Datentypen und zugehörige Prozeduren zur Verfügung zu stellen (*Bild 5*).
Ein Anwender kann nun den neuen Datentyp genauso wie die Standardtypen verwenden und eigene Programme damit entwerfen (*Bild 6*). Die Erzeugung neuer Datentypen sowie die Möglichkeit, Prozeduren bereitstellen zu können, unterstützt das modulare Arbeiten. Ein Paket der oben gezeigten Form kann im System leicht insertiert (vorübersetzt und eingebunden) werden. Alle Features des Paketes können anschließend allen Benutzern zugänglich gemacht werden. Diese Methode führt uns nun in die Tiefen des Eumel-Betriebssystems.

```
REAL PROC integral (REAL PROC (REAL CONST) f, REAL CONST unten, oben):
  REAL VAR summe :: 0.0;
  INT  VAR itstufe :: 1;
  REP
    REAL VAR altintegral :: summe,
             h           :: (oben - unten)/(10.0*real (itstufe)),
             summe 1     :: 0.0,
             summe 2     :: 0.0;
    INT VAR i;
    FOR i FROM 1 UPTO 10*itstufe - 1 REP
      IF (i MOD 2) = 1
         THEN summe 1 INCR f (unten + h * real (i))
         ELSE summe 2 INCR f (unten + h * real (i))
      FI
    PER;
    summe := h/3.0 * (f (unten) + f (oben) + 2.0 * summe 2 + 4.0 * summe 1);
    itstufe INCR 1;
    put ("                    "); put (summe);line
  UNTIL round (altintegral, 7) = round (summe, 7) PER;
  summe
END PROC integral;

REAL PROC f (REAL CONST x):
  x*x*x*x + 2.0 * x
END PROC f;

put (integral (PROC sin, 0.0, pi )); line;
put (integral (PROC f  , 0.0, 1.0));
```

Bild 3. Prozeduren können als Parameter übergeben werden

Das Eumel-Betriebssystem

Das Eumel-Betriebssystem ist ein Multiuser-Multitasking-System. Auf der Hardwareebene besteht das System aus einem Rechnerkern, an dem über mehrere Kanäle Terminals, Drucker, Plotter, Diskettenlaufwerke o. ä. quasi-parallel arbeiten können. Dazu laufen im Computer nebeneinander verschiedene Prozesse ab, die in einem Zeitscheibenverfahren nacheinander bedient werden. Diese Prozesse werden „Task" genannt. So gibt es eine Task, die die Druckausgabe über einer Warteschlange verwaltet; eine weitere Task organisiert den Diskettenzugriff. Tasks existieren für Netzwerke, für Terminalverwaltung und andere Systemaufgaben. Daneben kann jeder Benutzer des Systems eine eigene Task (oder auch mehrere) für sich neu kreieren, die zum Arbeiten jederzeit an ein beliebiges Terminal angekoppelt und wieder abgekoppelt werden kann. Das gesamte System besteht aus mehreren Tasks (*Bild 7*).

Hier sind momentan drei Benutzertasks eingerichtet worden, die mit den Namen „Meier", „Krause" und „Schulze" angesprochen werden. Somit hat jeder Benutzer einen eigenen Arbeitsbereich samt Dateien und Prozessen, auf die andere Benutzer keinen oder nur kontrollierten Zugriff haben. Die Arbeitsweise mit Eumel kann man am besten anhand einer Beispielsitzung demonstrieren. Schaltet man den Computer an, so wird man nach Tastendruck (F1) mit dem

„SUPERVISOR“ verbunden. Dieser meldet sich mit der Zeile:

gib supervisor kommando:

Durch das Kommando

begin („meine Task“)

wird eine neue Task mit dem Namen „meine Task“ erzeugt. Das Tasksystem hat jetzt die Form

```
SUPERVISOR
    SYSUR
        T1
        ARCHIVE
        SCHEDULER
        configur
        OPERATOR
        PRINTER
UR
    PUBLIC
        Meier
        Krause
        Schulze
        meine Task
```

Die Task wird dann an das Terminal gekoppelt und meldet sich mit

gib kommando:

Hier nun kann man mit dem Editor ein Elan-Programm erstellen oder auch einen Brief schreiben:

edit („meine Datei“)

Handelt es sich um ein Elan-Programm, kann man dieses übersetzen und gleich ausführen lassen mit:

run („meine Datei“)

Ein Programmlisting (oder den Ausdruck des Briefes) erhält man mit

print („meine Datei“).

Im System befindet sich eine Task „PRINTER“, die alle Druck-Aufträge in einer Warteschlange sammelt und sie nacheinander dem Drucker (über einen Druckertreiber) liefert. Der Benutzer kann somit während des Druckvorgangs weiterarbeiten. Mitgeliefert werden Druckertreiber für die z. Zt. gängigen Drucker.

Eine Nicht-Elan-Programmdatei kann selbstverständlich ebenfalls editiert und gedruckt werden. Der Editor stellt dazu komfortable Kommandos zum Textbearbeiten (Suchen, Ersetzen, Markieren) und Befehle zur Druckkosmetik bereit. Hervorstechende Eigenschaft des Editors ist die „Lernfähigkeit“: der Benutzer kann ganze Textpassagen oder auch Kommandos auf eine Taste legen.

Nach erfolgreicher Arbeit wird die Datei auf der Diskette abgelegt und die Task verlassen.

save („meine Datei“, archive) (Schreiben auf Diskette *)*

break (Verlassen der Task, Task bleibt im System erhalten *)*

end (Task wird beendet *)*

Der Computer kann jetzt abgeschaltet werden. Die Task „meine Task“ befindet sich auch nach Neustart (sofern nicht mit dem Kommando „end“ beendet) mit all den Dateien im System.

Tasks und auch Dateien können mit Passwörtern gegen unerlaubten Zugriff geschützt werden. Dateien können von einer Task zu einer anderen gesendet werden.

Die Besonderheit hinsichtlich der Multiuser-/Multitasking-Fähigkeit von Eumel lernt man schätzen, wenn man sich die hierarchische Struktur der Tasks genauer betrachtet. Das soll an einem Beispiel erläutert werden.

Ein Benutzer erzeugt eine Task mit dem Namen „WIDERSTAND“. In dieser Task wird das Programmpaket „PACKET widerstand“ (*Bild* 5) entwickelt und nach ausführlichen Tests in dieser Task eingebunden (in Eumel-Sprechweise „insertiert“). Dieser Vorgang umfaßt das Compilieren sowie das Einbinden des Compilats in das System. In der Task „WIDERSTAND“ scheint das Betriebssystem jetzt erweitert zu sein um die Befehle, die in der Paketschnittstelle aufgeführt sind („PARALLEL“, „SERIELL“, „put“ und „get“) sowie um den neuen Datentyp „WIDERSTAND“. Jetzt kann man z. B. auf der Kommandoebene damit arbeiten, z. B.:

<— gib kommando:

—> WIDERSTAND VAR w1, w2; get (w1), get (w2); put (w1 PARALLEL w2)

Damit sind die Möglichkeiten jedoch noch nicht erschöpft. Man kann jetzt die Task „WIDERSTAND“ zu einer sog. Managertask machen

—> global manager

Jetzt kann man eine neue Task (in Eumel-Sprechweise eine Sohntask) erzeugen:

<— gib supervisor kommando

—> begin („Anwender“, „WIDERSTAND“)

```
UR
    PUBLIC
        WIDERSTAND
            Anwender
        Meier
        :
```

In der Task „Anwender“ kann man nun z. B. das in *Bild 6* gezeigte Anwender-

```
WIDERSTAND VAR r1, r2;
put ("Bitte die Daten des ersten  Widerstandes :");
get (r1);
put ("Bitte die Daten des zweiten Widerstandes :");
get (r2);
put ("Serienschaltung     :"); put (r1 SERIELL r2); line;
put ("Parallelschaltung   :"); put (r1 PARALLEL r2)
```

Bild 6. Einmal definierte Datentypen können immer wieder eingesetzt werden

```
PACKET widerstand DEFINES WIDERSTAND,  (* neuer Datentyp für den *)
                                       (* Wechselstromwiderstand *)
                          PARALLEL,
                          SERIELL,
                          put,
                          get:

TYPE WIDERSTAND = COMPLEX;

REAL PROC abs (WIDERSTAND CONST w):
  ABS (CONCR (w))
END PROC abs;

REAL PROC dphi (WIDERSTAND CONST w):
  dphi (CONCR (w))
END PROC dphi;

WIDERSTAND OP + (WIDERSTAND CONST e,z):
  WIDERSTAND : (CONCR (e) + CONCR (z))
END OP +;

WIDERSTAND OP * (WIDERSTAND CONST e,z):
  WIDERSTAND : (CONCR (e) * CONCR (z))
END OP *;

WIDERSTAND OP / (WIDERSTAND CONST e,z):
  WIDERSTAND : (CONCR (e) / CONCR (z))
END OP /;

PROC put (WIDERSTAND CONST w):
  put (text (abs (w)) + " OHM / " + text (dphi (w)) + " Phasenversch.")
END PROC put;

PROC get (WIDERSTAND VAR w):
  put ("Ohmscher Widerstand :"); REAL VAR ohm; get (ohm);
  put ("Kapazit. Widerstand :"); REAL VAR kap; get (kap);
  CONCR (w) := complex (ohm, -kap)
END PROC get;

WIDERSTAND OP PARALLEL (WIDERSTAND CONST eins, zwei):
  (eins * zwei) / (eins + zwei)
END OP PARALLEL;

WIDERSTAND OP SERIELL (WIDERSTAND CONST eins, zwei):
  eins + zwei
END OP SERIELL

END PACKET widerstand
```

Bild 5. Pakete umfassen mehrere Prozeduren

```
SUPERVISOR                  (* Systemwart                              *)
    SYSUR                   (* Systemdienste                           *)
        T1                  (* Terminal-Wart                           *)
        ARCHIVE             (* Archivieren von Dateien auf Diskette    *)
        SCHEDULER           (* Zeit-und Prioritätenverwaltung          *)
        configurator        (* Schnittstellen- Anpassungen             *)
        OPERATOR            (* Systemdienste                           *)
        PRINTER             (* Warteschlangenverwaltung des Druckers   *)
                            (* Druckausgabe *)
UR                          (* Wurzel des Benutzerbereiches            *)
    PUBLIC                  (* Verwaltung der Benutzertasks            *)
        Meier               (* Benutzertask                            *)
        Krause              (* Benutzertask                            *)
        Schulze             (* Benutzertask                            *)
```

Bild 7. Taskstruktur eines Eumel-Systems

programm editieren und compilieren. Alle Tasks „unter“ der Task „WIDERSTAND“ sind nun privilegiert, diese Quasisystemerweiterung zu benutzen.

Die Auslieferung des Systems

Das Eumel/Elan-Paket für den Atari ST wird mit einem umfangreichen deutschen Handbuch, mehreren Disketten und einem ROM-Cartridge geliefert. Eumel setzt einen Atari ST mit 1 MByte RAM und einer Festplatte voraus; es gibt auch Demoversionen für kleinere Konfigurationen. In der ROM-Cartridge ist die Eumel/Elan-Treibersoftware für den Betrieb der Festplatte enthalten. Auf den Disketten sind im wesentlichen Utilities, Druckertreiber und Anwenderprogramme untergebracht. Neben einem LISP- und einem PROLOG-Interpreter, einer sehr leistungsfähigen Textverarbeitung und einem Basic-Compiler wird ein Programm angeboten, das zur Simulation kontinuierlicher Vorgänge dient (DYNAMO). Der LISP-Interpreter und der PROLOG-Interpreter sind, wie auch die Programmierhilfe für Anfänger (Hamster genannt), kostenlos. Anwendersoftware wie etwa Finanzbuchhaltung, Teletext oder ein Oberstufen-Verwaltungsprogramm und ein interaktives Stundenplanprogramm sind unter Eumel verfügbar. Auch zum Lesen oder Schreiben von MS-DOS-Dateien ist ein entsprechendes Programm erhältlich.

Literatur

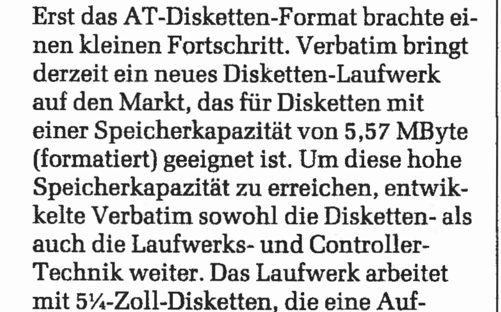
[1] *Klingen/Liedtke:* Elan in 100 Beispielen, B.G. Teubner, Stuttgart 1985.

[2] *Hommel, G.:* Elan-Sprachbeschreibung, Akademische Verlagsgesellschaft, Wiesbaden 1979.

[3] *Klinger/Liedke:* Programmierer in ELAN, B. G. Teubner, Stuttgart 1983.

[4] *Danckwerts, Vogel, Bovermann:* Elementare Methoden der Kombinatorik, B. G. Teubner, Stuttgart 1985.

Speicherriese für PCs



Die MS-DOS-Welt hinkte in ihren Disketten-Speicherkapazitäten lange Zeit dem technisch Realisierbaren hinterher. Erst das AT-Disketten-Format brachte einen kleinen Fortschritt. Verbatim bringt derzeit ein neues Disketten-Laufwerk auf den Markt, das für Disketten mit einer Speicherkapazität von 5,57 MByte (formatiert) geeignet ist. Um diese hohe Speicherkapazität zu erreichen, entwikkelte Verbatim sowohl die Disketten- als auch die Laufwerks- und Controller-Technik weiter. Das Laufwerk arbeitet mit 5¼-Zoll-Disketten, die eine Aufzeichnungsdichte von 384 TPI (tracks per inch) aufweisen. Eine herkömmliche MS-DOS-Diskette hat dagegen nur eine Aufzeichnungsdichte von 48 TPI. Die Zahl der Sektoren wurde bei den neuen Disketten auf 17 erhöht. Von Verbatim werden die Disketten bereits vorformatiert geliefert. Die logische Formatierung mit dem mitgelieferten Programm EKFORMAT dauert daher nur rund 8 s. Es ist erfreulich, daß das Laufwerk auch Disketten im AT-Format und im Standard-MS-DOS-Format lesen kann.
Die mitgelieferte Controller-Platine ist in konventioneller Technik aufgebaut. Als Controller-Baustein wird der weitverbreitete FDC 765 verwendet. Einige PAL-Bausteine übernehmen die Funktionen, die der FDC 765 nicht kann. Die Controller-Platine ist eine PC-Einsteckkarte in halber Baulänge. Der Verbatim-Controller kann sowohl mit einem bereits im System vorhandenen Floppy-Controller zusammenarbeiten oder diesen auch ersetzen. Er kann auch Laufwerke mit 40 Spuren bedienen. Mit DIP-Schaltern wird die gewünschte Betriebsart eingestellt. Bis zu vier Laufwerke steuert der Controller gleichzeitig.
Das neue Speichersystem wird komplett mit Treiber-Software und dem Installations-Handbuch geliefert. Derzeit ist das Handbuch jedoch nur in Englisch erhältlich.
Während unseres Tests arbeitete der Verbatim-Controller gleichzeitig mit einem anderen Floppy-/Festplatten-Controller im System. Der „Kabelsalat“ nimmt bei zwei Contollern im System leicht besorgniserregende Größen an: Die Verlegung der bereits konfektionierten Kabel muß sehr sorgfältig durchgeführt werden, damit diese nicht den Kühlluftstrom blockieren und es im System zu einem Wärmestau kommt. Das Laufwerk erwärmt deutlich das Innere eines PCs.

Interessant ist das Verbatim-System als Backup-Medium für Festplatten. Um den Inhalt einer 20 MByte-Festplatte zu sichern werden nur vier Disketten benötigt. Im Gegensatz zu Magnetbändern kann auf Programme und Daten sofort zugegriffen werden. Leider kann jedoch nicht von diesen Disketten gebootet werden, da MS-DOS dieses Format noch nicht kennt. Folgende MS-DOS-Befehle funktionieren nicht: FORMAT, SYS, DISKCOPY, DISKCOMP.

Hingegen funktionieren COPY, XCOPY und CHKDSK wie gewohnt. Das mitgelieferte Formatier-Programm EKFORMAT wird mit der Anweisung DEVICE = EKDRIVER.SYS in die CONFIG.SYS-Datei eingebunden. *St*



5,57 MByte auf einer Diskette abspeichern

Ulrich Cebulla

256k-EGA mit variabler Bildgröße

EGA mit monochromen Bildschirmen

Die 256k-EGA ist für monochrome Bildschirme genauso geeignet, wie für Farbbildschirme. Anwender, die nur monochrome Grafiken bearbeiten, müssen nicht mit einer weniger leistungsfähigen Adapterkarte wie z. B. der Hercules Karte vorlieb nehmen.

Ein Einwand gegen den EGA-Einsatz mit monochromen Bildschirmen ist zunächst richtig, daß nämlich jeder Besitzer einer 256k-EGA mit monochromem Bildschirm eine „Speicherleiche" von 192 KByte auf seiner EGA-Karte hat. Wie man diesen Speicher für den monochromen Bildschirm zum Leben erweckt und was man dann mit den 256 KByte der EGA machen kann (mit Monochrom- oder Farbbildschirm), wird in diesem Artikel näher erläutert.

Die Ursache dafür, daß bei Verwendung eines monochromen Bildschirmes an der EGA nur die ersten 64 KByte erreichbar sind, liegt an der Initialisierung des Attribute Controllers. Hier macht das BIOS der IBM-EGA aus gutem Grund keinen Unterschied zwischen einer 64k-EGA und einer IBM-EGA mit mehr als 64 KByte. Daß die gleiche Initialisierung allerdings auch bei einigen „Clone"-EGAs mit 256 KByte Verwendung findet, ist ein Fehler. Um die freibleibenden 192 KByte der EGA auch benutzen zu können, muß hauptsächlich das „Color Plane Enable Register" des Attribute Controllers entsprechend gesetzt werden. Dann stehen auch alle Palette-Register zur Verfügung, und damit außer den bekannten Farben 0, 3, C und F auch alle Farben von 0...F. Das Beispielprogramm (*Bild 1*) zeigt, wie man das hervorragend nutzen kann.

Bevor wir uns näher mit dem Programm befassen, wenden wir uns noch ein paar grundsätzlichen Betrachtungen zu.

Daß der Adreßbereich und damit der Speicherbedarf für eine Bildschirmseite im Grafikmodus 28000 Byte für jede der vier Bildspeicherebenen C0...C3 beträgt, also insgesamt 112 KByte, ist wohl den meisten EGA-Anwendern bekannt. Was kann man nun mit den restlichen 144 KByte alles anstellen?

Eine naheliegende Antwort auf diese Frage ist sicher die, zwischen zwei gleichzeitig im Speicher vorhandenen Bildschirmseiten hin und her zu schalten. Das kann mit einer vom BIOS bereitgestellten Funktion erreicht werden; was jedoch nur sehr selten genutzt wird. Möchte man im Textmodus vier verschiedene Zeichensätze gleichzeitig im EGA-Speicher halten, sind 256 KByte erforderlich. Für grafische Anwendungen sind jedoch noch eine ganze Reihe von meist unbekannten Möglichkeiten der Bildschirmspeichernutzung vorhanden.

Auf eine Anwendung, die variable Bildgröße, wird hier näher eingegangen. Nennen wir sie einfach EGA-VAR-BILD (variable Bildgröße), mit einer variablen Bildgröße von 640 × 350 über 640 × 819 bis 1488 × 352, wobei das kleinste Bild mit 640 × 350 das sogenannte normale Bild ist. *Bild 2* zeigt ein Bild in der Größe 912 × 570. Selbstverständlich ist das sichtbare Bildfenster immer nur das normale Bild (640 × 350 Punkte oder 4 × 28 KByte = 112 KByte).

Um den restlichen Bildschirmspeicher von 144 KByte sichtbar zu machen, muß nur das Bildfenster sowohl horizontal als auch vertikal bewegt werden können. Wie so etwas mit einer EGA-Karte funktioniert und wie noch ein paar andere brauchbare Grafikfunktionen implementiert werden, zeigen wir an einem Programmbeispiel.

Zuerst jedoch einige grundsätzliche Bemerkungen zu den Bildschirmen und zu dem Bildschirmadapter EGA. Die Auflösung eines Bildschirmes – sowohl horizontal als auch vertikal – ist bei den meisten Bildschirmen konstruktionsbedingt und auch nur manchmal begrenzt variabel, während die Auflösung des Bildschirmadapters meist einen größeren Spielraum aufweist.

Bei den Enhanced Color Displays ist die Auflösung in der Regel 320 × 200, 640 × 200 und 640 × 350. Bei der IBM-EGA-Karte ist die vertikale Auflösung programmierbar bis maximal 1024 Scan-Zeilen und horizontal bis maximal 2048 Bildpunkte (den Rücklauf mit eingeschlossen). In mc 1...4/87 wurde näher darauf eingegangen. Von der Speicherorganisation her gesehen, kann eine horizontale Scan-Zeile sogar maximal 4096 Punkte (512 Byte) lang sein.

Weiterhin ist der CRT-Controller der EGA – von der Bildfläche aus betrachtet – in der Lage, 64 KByte (vier Speicherebenen mit je 64 KByte) zu adressieren. Das bedeutet, daß im EGA-Speicher ein Bild mit einer maximalen Auflösung von 65 535 × 8 = 524 280 Punkten aufgebaut werden kann. Um ein solches Bild verarbeiten zu können, ist es erforderlich, das Bildfenster über den gesamten Bildspeicher bewegen zu können. Wie das in vertikaler Richtung geschieht, wurde bereits in mc 4/87 beschrieben.

Es werden dazu im wesentlichen nur die beiden CRT-Controller-Register „Start Adress High" und „Start Adress Low" benutzt. Für die horizontale Bewegung des Bildfensters muß zusätzlich noch ein CRT-Controller-Register herangezogen werden, und zwar das Offset-Register mit der Portadresse 3B5H.13 (für monochrome Bildschirme) und 3D5H.13 (für Enhanced Color Bildschirme).

Im Grafikmodus wird in diesem Register die logische Länge einer Scan-Zeile abgelegt, und zwar bei der 256k-EGA in „WORD" (1 word = 2 Byte). Das sind im Normalfall 28H = 80 Byte = 640 Punkte. Das Ende einer horizontalen Zeile liegt deshalb immer auf einer 2-Byte-(16-Punkte-)Grenze.

Selbstverständlich sollte dieses Register nie einen Wert, der kleiner als 28H ist (also weniger als 640 Punkte horizontal), enthalten.

Für den CRT-Controller bedeutet das, daß am Ende einer jeden Scan-Zeile (horizontal retrace) der Inhalt des „Offset Registers" (in Byte), auf das Start Adress High/Low-Register addiert wird. Diese Adresse gibt die Anfangsadresse der nächsten Scan-Zeile an, ab welcher

```
{****************************************************************************}
{*                            I B M   E G A                                 *}
{*                       Turbo Pascal Grafik-Programm                       *}
{*  E G A   D R A W      (CAD mit der IBM EGA)                              *}
{*  Ulrich Cebulla                                                          *}
{*  7036 Schoenaich, Hermann Werner Str.6                    01.05.87       *}
{****************************************************************************}

Program EGA_TCAD;
{$I grafik4.ega}           { IBM EGA Unterstuetzungs-Programme }
{$I print.ega}             { Bildschirm drucken }
{$I funktion.cad}          { Kurzbeschreibung der Funktions-Tasten }
type namestring=string[8];
     bspeicher=array [0..32767] of byte;
     bildpointer=^bspeicher;
var
     bildstart:bildpointer;
     bildfile: file of bspeicher;
     bildvh: file of integer;
var
   x,y,x1,y1:integer;           { x/y Koordinaten }
   xline,yline:integer;         { x/y-Position fuer drawline }
   xx1:array[0..3] of integer;  { window array x links oben }
   xx2:array[0..3] of integer;  { window array x rechts unten }
   yy1:array[0..3] of integer;  { window array y links oben }
   yy2:array[0..3] of integer;  { window array y rechrs unten }
   scrollsw:boolean;          { scroll Schalter }
   savestart:integer;       { Startzeile }
   savestacth:integer;      { save Startzeile (help) }
   savehorpos:integer;      { save horizontale Position }
   savevertpos:integer;     { save vertikale Position }
   sr:byte;                 { schreiben/radieren }
   draw,savedraw,xy,help,fontsw,text,invertext,vertext:boolean;
   speed,savespeed:byte;  { Fadenkreuz Geschwindigkeit }
   fdkz,plane,saveplane,mode:byte;   { Fadenkreuz-/Zeichnung-Speicherebene }
   bild:byte;               { Bildebene }
   readebene:byte;          { Auswahl der zu lesenden Bildspeicherebene. }
   linebuff:integer;        { Freie Bildschirmzeile }
   taste,texttaste,ft:char;
   egamax  :boolean;
   v1,v2,v3,v4,v5:real;
{---------------Test der x/y-Eingabe, innerhalb von 256k---------------------}
Procedure egamem;
begin
v1:=horpel+1;
v2:=vert+2;               { 1 scan line fuer die move procedure }
v3:=32767;
v4:=16;
v5:=v3*v4+8;              { max EGA-Speicher 65535 Bytes/Plane }
                          { = 524280pel - horpel               }
if v1*v2 > v5
then begin
     gotoxy(50,2);  write(horpel+1);
     gotoxy(50,3);  write(vert+1);
     gotoxy(27,4); write('EGA-Speicher  = ',(v1*v2):7:0,' pel, EGA-Ueberlauf');
     gotoxy(33,5); write('max pel = ',v5:7:0,' - ',v1:4:0,' = ',v5-v1:7:0);
     gotoxy(34,6); write('oder y = ',(v5/v1-1):4:0);
     halt;
     end;

if v1*(v2-1) > v5/2 then egamax:=true            { Bild > 32k }
                    else egamax:=false;
end;
{---------------Fadenkreuz set/reset-----------------------------------------}
procedure fadenkreuzsr(xx,yy:integer;map,setreset:byte);
begin
             Speicherebene(map);
             y:=yy;
             for x:=xx-3 to xx+3
                 do pset(x,y,setreset,0);    { Fadenkreuz hor. set/reset }
             x:=xx;
             for y:=yy-3 to yy+3
                 do pset(x,y,setreset,0);    { Fadenkreuz vert. set/reset }
end;
{---------------Zeichenmodus anzeigen---------------------------------------}
Procedure display;
begin
     Speicherebene(mode);
     gotoxy(23,35);
     if draw
     then
          write('+ zeichnen ')
     else
          write('+ bewegen  ');
     if sr=15                      { Wenn "sr" (set/reset) = 15 (Farbe),  }
     then                          { dann wird gezeichnet, sonst gloescht  }
          write('3','S','3 ')
     else
          write('3','R','3 ');

     write(speed,' Punkte                 ');
end;
{---------------Anzeige der x/y position des Fadenkreuzes-------------------}
procedure displayxy;
begin
   Speicherebene(mode);
   if xy                                { x/y Werte anzeigen }
   then begin
        gotoxy(3,31); Write(x1,'   ');        { x-Wert horizontal anzeigen}
        wrthor:=false;
        gotoxy(1,30); Write(y1,'  ');        { y-Wert vertikal anzeigen}
        wrthor:=true;
        end
   else begin
        gotoxy(3,31); Write('    ');   { x-Wert horizontal anzeigen}
        wrthor:=false;
        gotoxy(1,30); Write('   ');   { y-Wert vertikal anzeigen}
        wrthor:=true;
        end;
end;
{---------------Auswahl der Zeichenebene (C0, C1, C0 und C1)---------------}
procedure planeselect(select:byte);
var taste:char;
begin
     Speicherebene(mode);
     plane:= select;
     bild:= plane + mode + fdkz;
     gotoxy(1,35);
     write('Plane: ');
     if plane = 1 then write('C0.   ');
     if plane = 2 then write('C1.   ');
     if plane = 3 then write('C0+C1.');
     Bildebene(bild);
end;
{---------------Fadenkreuz Bewegen / Bild zeichnen--------------------------}
Procedure fadenkreuz(lr,ud:integer);
var speedloop:integer;
begin
   if not help
   then begin
        if not draw
        then
            speedloop:=1                  { schnelle loop }
        else
            speedloop:=speed;             { pel fuer pel loop }

        i:=1;
        repeat
           {================zeichnen===================}
if draw
then begin
     speicherebene(plane);       { Speicherebene der Zeichnung }
     pset(x1,y1,sr,0);           { Bild zeichnen  }
     end;
{============================================}

fadenkreuzsr(x1,y1,fdkz,0);      { altes Fadenkreuz loeschen }

if draw
then begin
     x1:=x1+lr;                  { neue x-Position, langsam }
     y1:=y1+ud;                  { neue y-Position, langsam }
     end
else begin
     x1:=x1+(lr*speed);          { neue x-Position, schnell }
     y1:=y1+(ud*speed);          { neue y-Position, schnell }
     end;
```

Bild 1. Das Programm EGA-TCAD, mit dem auch umfangreiche Grafiken sehr schnell erstellt werden

```
        if x1 > horpel         { Randbegrenzung rechts }
        then begin
             x1:=horpel;
             i:=speedloop;     { loop Ende }
             end;
        if x1 < 0              { Randbegrenzung links }
        then begin
             x1:=0;
             i:=speedloop;     { loop Ende }
             end;

        if y1 > vert           { Randbegrenzung unten }
        then begin
             y1:=vert;
             i:=speedloop;     { loop Ende }
             end;
        if y1 < 0              { Randbegrenzung oben }
        then begin
             y1:=0;
             i:=speedloop;     { loop Ende }
             end;
        fadenkreuzsr(x1,y1,fdkz,15);      { neues Fadenkreuz setzen }
        if xy and not text
        then
           displayxy;
      i:=i+1;
      until i > speedloop;
      end;
end;
{---------------Fadenkreuz finden (oben, links)----------------------------}
Procedure findfadenkreuz;
begin
  fadenkreuzsr(x1,y1,fdkz,0);      { altes Fadenkreuz loeschen }
  x1:=hscrollpos*8 + 3;
  y1:=vscrollpos+3;
  fadenkreuzsr(x1,y1,fdkz,15);     { neues Fadenkreuz setzen }
  displayxy;
end;
{---------------Bildspeicher Modus (mode) / Fadenkreuz (fdkz) initialisieren }
Procedure init2;
begin
   graftext:=false;                 { normaler Text }
   speicherebene(fdkz);             { Speicherebene Fadenkreuz }
   fillchar(mem[$A000:0],-1,0);     { Speicher loeschen }
   fadenkreuz(0,0);
   mode:=8;                         { Speicherebene Modus }
   Speicherebene(mode);
   fillchar(mem[$A000:0],-1,0);     { Speicher loeschen }
   gotoxy(15,35);
   wrtinvert:=true;
   write(' Modus:');
   wrtinvert:=false;
   display;
end;
{---------------Bildspeicher auf Diskette speichern------------------------}
procedure SaveScreen(filename,filemode:namestring);
begin
   assign(bildfile,concat(filename,filemode));
   {$I-} rewrite(bildfile); {$I+}
   if ioresult = 0
   then begin
        bildstart:=ptr($A000,000);
        write(bildfile,bildstart^);
        bildstart:=ptr($A800,000);
        if egamax
        then
           write(bildfile,bildstart^);
        close(bildfile);
        end
   else begin
        Speicherebene(mode);
        gotoxy(23,34);
        write('IO error file ',concat(filename,filemode),' schreiben');
        end;

   assign(bildvh,concat(filename,filemode,'v'));
   {$I-} rewrite(bildvh); {$I+}
   if ioresult = 0
   then begin
        write(bildvh,vert,horoffset);
        close(bildvh);
        end
   else begin
        Speicherebene(mode);
        gotoxy(23,34);
        write('IO error file ',concat(filename,filemode),' schreiben');
        end;
  end;
{---------------Bildspeicher von der Diskette laden------------------------}
procedure LoadScreen(filename,filemode:namestring);
begin
   assign(bildvh,concat(filename,filemode,'v'));
   {$I-} reset(bildvh); {$I+}
   if ioresult = 0
   then begin
        Bildebene(0);
        read(bildvh,vert,horoffset);
        close(bildvh);

        setcrtoffset;
        egamem;
        startzeile:=0;              { Startzeile des Bildschirmes }
        savestart:=0;
        CRTstartaddr(startzeile); { Startzeile 0000 setzen }


        assign(bildfile,concat(filename,filemode));
        {$I-} reset(bildfile); {$I+}
        if ioresult = 0
        then begin
             bildstart:=ptr($A000,000);
             read(bildfile,bildstart^);
             bildstart:=ptr($A800,000);
             if egamax
             then
                 read(bildfile,bildstart^);
             close(bildfile);

             init2;
             Planeselect(plane);
             end
        else begin
             Speicherebene(mode);
             gotoxy(23,34);
             write('IO error file ',concat(filename,filemode),' lesen');
             end;
        end
   else begin
        Speicherebene(mode);
        gotoxy(23,34);
        write('IO error file ',concat(filename,filemode),' lesen');
        end;
end;
{---------------Helpbild auf dem Bildschirm anzeigen-----------------------}
procedure helpbild;
var a:byte;
begin
   help:=not help;
   if help
   then begin
        saveplane:=plane;
        savestarth:=startzeile;     { alte Startzeile (sroll) halten }
        startzeile:=0;
        CRTstartaddr(startzeile);  { Startzeile 0000 setzen }
        Speicherebene(fdkz);
        Bildebene(fdkz+mode);
        helptext(horpel,vert);
        end
   else begin
        fillchar(mem[$A000:0],-1,0);
        startzeile:=savestarth;
        CRTstartaddr(startzeile);    { alte Startzeile in "scroll" Proc. laden}
        fadenkreuzsr(x1,y1,fdkz,15); { Fadenkreuz setzen }
        planeselect(saveplane);
        end;
end;
{---------------Modus anzeigen, wenn nicht sichtbar------------------------}
procedure modus;
begin
   if scrollsw
   then begin
        savestart:=startzeile;      { alte Startzeile (sroll) halten }
        savevertpos:=vscrollpos;
        savehorpos:=hscrollpos;
```

```
        vscrollpos:=0;
        hscrollpos:=0;
        startzeile:=0;
        CRTstartaddr(startzeile);  { Startzeile 0000 setzen }
        scrollsw:=false;
        end
   else begin
        startzeile:=savestart;
        vscrollpos:=savevertpos;
        hscrollpos:=savehorpos;
        CRTstartaddr(startzeile);  { alte Startzeile aus "scroll" Proc. laden}
        scrollsw:=true;
        end;
end;
{---------------Text in der Zeichnung-------------------------------------}
Procedure textproc;
var xt,yt:integer;
begin
   if not help
   then begin
        savedraw:=draw;               { Zeichnen-Modus festhalten }
        draw:=false;                  {                           }
        savespeed:=speed;             {                           }

        speed:=1;
        text:=true;                   { Textmodus ein }

        Speicherebene(mode);
        gotoxy(23,35);                { Modus loeschen }
        write('                                  ');
        gotoxy(23,35);
        write('Text, ');              { Modus in Zeile.35 setzen }

        if fontsw                     { 8x14 / 8x8 Font }
        then if clrbox
             then
                 write('8x14 Box')
             else
                 write('8x14')
        else if clrbox
             then
                 write('8x10 Box')
             else
                 write('8x8');

        write(', SF=',size);

        if invertext                  { invertiert schreiben }
        then begin
             wrtinvert:=true;         { siehe Procedure  "writechar" }
             write(', invert ');
             end;

        if vertext                    { vertikal schreiben }
        then begin
             write(', vertikal');
             wrthor:=false;
             end;

        graftext:=true;

        xt:=x1;

        if y1 > (vert-box)
        then begin
             yt:=vert-box+1;
             fadenkreuz(0,-(y1-yt));  { Fadenkreuz y-max setzem }
             end
        else
            yt:=y1;                   { y-Ache max vert - box }

        gotoxy(xt,yt);                { 1. x/y-Position des Textes setzen }

        if fontsw                     { Text-Font Auswahl }
        then
           fontpointer(false);        { 8x14 font }
        {==================Text-Loop========================}

        repeat
             read(kbd,texttaste);     { Taste lesen        }
             speicherebene(plane);    { Speicherebene Zeichnung}

             write(texttaste);        { Charakter schreiben }
                          { Fadenkreuz im Textmodus }
             if nextxy and not (texttaste = chr(13))
             then begin
                  if texttaste = chr(8)       { BS ? (Backspace)}
                  then if wrthor
                       then
                          fadenkreuz(-8*size,0)   { BS 1 Byte  <= }
                       else
                          fadenkreuz(0,-box*size) { BS 1 Char box nach oben }
                  else if wrthor
                       then
                          fadenkreuz(8*size,0)    { naechster Charakter => }
                       else
                          fadenkreuz(0,box*size); { naechster Char nach unten }
                  end;
        until texttaste = chr(13);  { CR = Text-Ende      }
        {==============================================}
        if fontsw
        then
           fontpointer(true);       { zurueck nach 8x8 font }

        graftext:=false;            { zurueck zum Zeichnen-Modus }
        wrtinvert:=false;
        wrthor:=true;
        draw:=savedraw;

        speed:=savespeed;
        display;
        text:=false;                { Textmodus aus }
        end;
end;
{---------------zeichnen/loeschen--------------------------------------------}
Procedure setrstproc;               { Punkt setzen oder`loeschen }
begin                               { Punkt setzen: "sr" = 15   }
     if sr = 0                      { Punkt loeschen:"sr" = 0    }
     then
        sr:=15                      { sr (set/reset) Farbe fuer bildzeichnen }
     else                           { in der Procedure "Fadenkreuz".       }
        sr:=0;
     display;
end;
{---------------Bildschirm auf Diskette schreiben/lesen-------------------}
Procedure diskbild(lesen:boolean);
type name = string[8];
var diskname:name;
    move    :byte;
begin
   scrollsw:=true;
   modus;
   Speicherebene(mode);
   gotoxy(23,34);
   write(                                   ');     { IO Error Zeile loeschen }
   if lesen
   then begin
        gotoxy(23,35);
        write('Diskette lesen                ');
        end
   else begin
        gotoxy(23,35);
        write('Diskette schreiben            ');
        end;
   gotoxy(65,35);
   if lesen
   then
        write('L_Name:        ')
   else
        write('S_Name:        ');
   gotoxy(73,35);

   diskname:='';
   readln(diskname);                     { Filename fuer Lesen/Schreiben }
   if diskname <> ''
   then begin

        if lesen                         { Diskette lesen }
        then begin
             case plane of
             1: begin                    { C0 }
                speicherebene(plane);    { Zeichnungs-Ebene C0 }
                loadscreen(diskname,'.C0'); { Bild von der Diskette laden }
                end;
             2: begin                    { C1 }
                speicherebene(plane);    { Zeichnungs-Ebene C1 }
```

```
              loadscreen(diskname,'.C1'); { Bild von der Diskette laden }
              end;
           3: begin                        { C0 & C1 }
              speicherebene(plane and 1); { Zeichnungs-Ebene C0 }
              loadscreen(diskname,'.C0'); { Bild von der Diskette laden }
              speicherebene(plane and 2); { Zeichnungs-Ebene C1 }
              loadscreen(diskname,'.C1'); { Bild von der Diskette laden }
              end;
           end;   { end case }
           end
       else begin                          { Diskette schreiben }
           case plane of
           1: begin                        { C0 }
              speicherebeneread(0);        { Zeichnungs-Ebene C0 }
              savescreen(diskname,'.C0'); { Bild auf Diskette speichern }
              end;
           2: begin                        { C1 }
              speicherebeneread(1);        { Zeichnungs-Ebene C1 }
              savescreen(diskname,'.C1'); { Bild auf Diskette speichern }
              end;
           3: begin                        { C0 & C1 }
              speicherebeneread(0);        { Zeichnungs-Ebene C0 }
              savescreen(diskname,'.C0'); { Bild auf Diskette speichern }
              speicherebeneread(1);        { Zeichnungs-Ebene C1 }
              savescreen(diskname,'.C1'); { Bild auf Diskette speichern }
              end;
           end;
           end;   { end case }
       end;
       scrollsw:=true;
   modus;
   display;
end;
{----------------Bildschirm nach oben/unten scrollen-------------------------}
procedure scroll(setstart:boolean);
var    taste:char;
     crtstart:integer;
begin
   Speicherebene(mode);
   gotoxy(23,35);
   write('scrollen                           ');
   scrollsw:=true;

   crtstart := startzeile;
   repeat
   if not setstart
   then begin
       read(kbd,taste);
       case ord(taste) of
       27: begin read(kbd,ft);             { ESC                         }
           case ord(ft) of

           72 : begin
                if vscrollpos < vert-349
                then begin
                     vscrollpos:=vscrollpos+1;
                     crtstart := crtstart + scrolline;  {Zeilen aufwaerts }
                     end;
                end;
           80 : begin
                if vscrollpos > 0
                then begin
                     vscrollpos:=vscrollpos-1;
                     crtstart := crtstart - (scrolline); { Zeilen abwaerts  }
                     end;
                end;
           75 : begin
                if hscrollpos < hor-80
                then begin
                     hscrollpos:=hscrollpos+1;
                     crtstart := crtstart + 1;          { 8 Punkte links }
                     end;
                end;
           77 : begin
                if hscrollpos > 0
                then begin
                     hscrollpos:=hscrollpos-1;
                     crtstart := crtstart - 1;          { 8 Punkte rechts }
                     end;
                end;

           end;   { end of case }
           end;
```

```
           end;       { end of case }
           end
     else
         taste:='C';             { wenn setstart true ist, nicht auf "C" warten.}

     CRTstartaddr(crtstart);      { neue Startadresse in CRT }

     startzeile := crtstart;    { Start Zeile speichern }
     until taste='C';           { Warteschleife, bis die "C"-Taste gedrueckt wird}

     display;
end;
{----------------Bild vertikal/horizontal im Bildspeicher bewegen----------}
Procedure movebild;
var i :integer;
begin
   Speicherebene(mode);
   gotoxy(23,35);
   write('move (hor/vert)                        ');

   EGAwritemode(1);          { 32 bit read/write Modus (write mode 1) }
   speicherebene(plane);

   linebuff:= (vert+1)*hor;  { nicht benutzter Speicher im Bildschirmspeicher }
                             { wird als buffer genutzt.                      }

{=====================Loop Bild horizontal/vertikal rotieren================}
repeat
  read(kbd,taste);
  case ord(taste) of
   27: begin read(kbd,ft);               { ESC                    }
       case ord(ft) of

       75:   for i:=0 to vert        { Bild horizontal 1/(2) Bytes nach links }
             do begin
                move(mem[egabase:(i*hor)],mem[egabase:linebuff],1);
                move(mem[egabase:(i*hor+1)],mem[egabase:(i*hor)],(hor-1));
                move(mem[egabase:linebuff],mem[egabase:(i*hor+(hor-1))],1);
                end;
       77:   for i:=0 to vert        { Bild horizonzal 1/(2) Bytes nach rechts}
             do begin
                move(mem[egabase:(i*hor+(hor-1))],mem[egabase:linebuff],1);
                move(mem[egabase:(i*hor)],mem[egabase:(i*hor+1)],(hor-1));
                move(mem[egabase:linebuff],mem[egabase:(i*hor)],1);
                end;
       72: begin                     { Bild vertikal eine scan line nach oben }
             move(mem[egabase:000],mem[egabase:linebuff],hor);
             move(mem[egabase:hor],mem[egabase:000],vert*hor);
             move(mem[egabase:linebuff],mem[egabase:(vert*hor)],hor);
             end;
       80: begin                     { Bild vertikal eine scan line nach unten}
             move(mem[egabase:(vert*hor)],mem[egabase:linebuff],hor);
             move(mem[egabase:000],mem[egabase:hor],vert*hor);
             move(mem[egabase:linebuff],mem[egabase:000],hor);
             end;
       end;  { end of case }
       end;
  end;       { end of case }

until taste = 'M';           { Loop mit Taste "M" verlassen }

EGAwritemode(0);             { zurueck zu write mode 0 }

display;
end;
{----------------Save x/y links oben----------------------------------------}
Procedure savex1y1(window:byte);
begin
     fadenkreuzsr(xx1[window],yy1[window],mode,0);  { Fadenkreuz reset }
     xx1[window]:=x1;
     yy1[window]:=y1;
     fadenkreuzsr(xx1[window],yy1[window],mode,15); { Fadenkreuz setzen }
end;
{----------------Save x/y rechts unten--------------------------------------}
Procedure savex2y2(window:byte);
begin
     fadenkreuzsr(xx2[window],yy2[window],mode,0);  { Fadenkreuz reset }
     xx2[window]:=x1;
     yy2[window]:=y1;
     fadenkreuzsr(xx2[window],yy2[window],mode,15); { Fadenkreuz setzen }
end;
{----------------Window kopieren (oder loeschen)-----------------------------}
Procedure copywindow(window:byte);
```

```
var cx,cy:integer;
begin
   speicherebene(mode);
   gotoxy(23,35);
   write('Fenster kopieren                 ');
   speicherebene(plane);      { Speicherebene der Zeichnung }
   cy:=y1;
   for y := yy1[window] to yy2[window]
   do begin
      cx:=x1;
      for x := xx1[window] to xx2[window]
      do begin
         pset(cx,cy,(pd(x,y,sr and plane,0)),0);   { copy read/write }
         cx:=cx+1;
         end;
      cy:=cy+1;
      end;
      display;
end;
{----------------x/y-Position fuer Linie zeichnen------------------------------}
Procedure xyline;
begin
     fadenkreuzsr(xline,yline,mode,0);  { Fadenkreuz reset }
     xline:=x1;
     yline:=y1;
     fadenkreuzsr(xline,yline,mode,15); { Fadenkreuz setzen }
end;
{----------------Linie zeichnen----------------------------------------------}
Procedure drawline;
var
   xlyo:boolean;
   lx1,lx2,ly1,ly2,xl,xr,yo,yu,xd,yd,lpx,lpy:integer;
   s,dx,dy:real;


begin
   lx1:=x1;
   lx2:=xline;
   ly1:=y1;
   ly2:=yline;
   if ((lx1 < lx2) and (ly1 < ly2)) or ((lx2 < lx1) and (ly2 < ly1))
   then
        xlyo:=true              { x links & y oben }
   else
        xlyo:=false;

   if lx1 < lx2                 { x links/rechts }
   then begin
        xl:=lx1;
        xr:=lx2;
        end
   else begin
        xl:=lx2;
        xr:=lx1;
        end;

   if ly1 < ly2                 { y links/rechts }
   then begin
        yo:=ly1;
        yu:=ly2;
        end
   else begin
        yo:=ly2;
        yu:=ly1;
        end;

   xd:=xr-xl;                           { x Differenz }
   yd:=yu-yo;                           { y Differenz }
   speicherebene(plane);
{....folgend, die 4 moegliche Situationen einer Linie....}
if xlyo
then begin
     if xd <= yd
     then begin
          s:=xd/yd;
          dx:=xl;
          for lpy:=yo to yu
          do begin
             lpx:=round(dx);
             pset(lpx,lpy,sr,0);    { Linie zeichnen }
             dx:=dx+s;
             end;
          end
     else begin
          s:=yd/xd;
          dy:=yo;
          for lpx:=xl to xr
          do begin
             lpy:=round(dy);
             pset(lpx,lpy,sr,0);    { Linie zeichnen }
             dy:=dy+s;
             end;
          end;
     end
else begin
     if xd <= yd
     then begin
          s:=xd/yd;
          dx:=xl;
          for lpy:=yu downto yo
          do begin
             lpx:=round(dx);
             pset(lpx,lpy,sr,0);    { Linie zeichnen }
             dx:=dx+s;
             end;
          end
     else begin
          s:=yd/xd;
          dy:=yo;
          for lpx:=xr downto xl
          do begin
             lpy:=round(dy);
             pset(lpx,lpy,sr,0);    { Linie zeichnen }
             dy:=dy+s;
             end;
          end;
     end;
end;
{----------------Bildspeicher drucken----------------------------------------}
Procedure printscr;              { Bildspeicherebene drucken }
begin
     gotoxy(23,35);
     Speicherebene(mode);
     write('Bildschirm drucken                  ');
     prtscr(plane,vert+1,horpel+1);      { Speicherebene C0/C1 drucken }
     gotoxy(23,35);
     write('                  ');
end;

{---------------------------------------------------------------------------}
Procedure clearplane;
begin
   Speicherebene(plane);
   fillchar(mem[$A000:0],-1,0);  { Speicher loeschen }
end;
{---------------------------------------------------------------------------}
{                            Program Start                                  }
{----------------------------var Initialisierung----------------------------}

begin
EGAmode:=0;                 { erlaube turbo gotoxy auaerhalb vom Graphicsmodus }
scrollsw:=true;
startzeile:=0;              { Startzeile des Bildschirmes }
savestart:=0;

x1:=320;                    { Startposition des Fadenkreuzes }
y1:=175;                    { in der Bildschirmmitte         }
for i:=0 to 3 do xx1[i]:=0;
for i:=0 to 3 do xx2[i]:=0; { alle Fixpunkte auf x/y=0 }
for i:=0 to 3 do yy1[i]:=0;
for i:=0 to 3 do yy2[i]:=0;
xline:=0;
yline:=0;
speed :=10;                 { cursor Geschwindigkeit 10 Punkte/ Tastendruck}
draw:=false;
xy:=false;
sr:=15;                     { Pset/Preset Farbe }
fontsw:=false;
invertext:=false;
vertext:=false;
text:=false;
fdkz:=4;                    { Fadenkreuz/Funktionen C2 }
bild:=5;                    { Bildebene: Zeichnung und Fadenkreuz}

clrscr;
delay(1);
```

```
{----------------Eingabe der Bildschirm x/y Werte---------------------------}
gotoxy(27,1);write('E G A - V A R - B I L D');
gotoxy(24,2); write('Horizontal (max x=1488) : ');
gotoxy(24,3); write('Vertikal   (max y= 818) : ');
gotoxy(50,2);
hor:=0;
readln(hor);
if hor = 0 then hor:=640;
if hor < 640 then hor:=640;
if hor > 1488 then hor:=1488;
horoffset:=hor div 16;
if hor <> horoffset*16 then horoffset:=horoffset+1;
gotoxy(50,3);
vert:=0;
readln(vert);
if vert = 0 then vert:=350;
if vert < 350 then vert:=350;
if vert > 818 then vert:=818;
vert:=vert-1;            { scan lines 0..vert }
hor := horoffset*2;      { Bytes, horizontal }
horpel:=hor*8-1;         { horizontal pel }

{----EGA Speichergrenze ueberschritten ??--}
egamem;      { EGA Memory test }
{------------nein, wenn hier--------------}


clrscr;
{--------------------------------------------------------------------------}
grafikEGA(true);            { Start Grafik }
grafiktext(true);           { Text in Grafik ein }
{------------------EGA/Bildschirm Auswahl-----------------------------------}

setcrtoffset;            { Zeilenlaenge ins CRT-Offset-Register }

if EGAmode = $0F         { Monochrom Display }
then begin
     blinking(false);      { blinken ausschalten }
     farbpalette($08,1);  { C0 }
     farbpalette($08,2);  { C1 }
     farbpalette($18,3);  { C0 & C1 }
     farbpalette($08,4);  { C2 }
     farbpalette($18,5);  { C0 & C2 }
     farbpalette($18,6);  { C1 & C2 }
     farbpalette($08,7);  { C0 & C1 & C2 }
     farbpalette($08,8);  { C3 }
     farbpalette($18,9);  { C0 & C3 }
     farbpalette($18,10); { C1 & C3 }
     farbpalette($18,11); { C0 & C1 & C3 }
     farbpalette($18,12); { C2 & C3 }
     farbpalette($08,13); { C0 & C2 & C3 }
     farbpalette($08,14); { C1 & C2 & C3 }
     farbpalette($18,15); { C0 & C1 & C2 & C3 }
     end
else begin               { Enhanced Color Display }
     farbpalette($24,1); { Zeichnung C0             : Farbe rot  }
                         { Zeichnung C1             : Farbe gruen }
     farbpalette($35,4); { Fadenkreuz/Funktion C2  : Farbe pink }
     farbpalette($3e,5); { Punktueberlappung C0/2   : Farbe gelb }
     farbpalette($3e,6); { Punktueberlappung C2/1   : Farbe gelb }
     farbpalette($3e,7); { Punktueberlappung C0/1/2 : Farbe gelb }
     farbpalette($3e,8); { Modus                    : Farbe gelb }
     farbpalette($39,12);{ Punktueberlappung        : Farbe blau }
     end;
{--------------------------------------------------------------------------}


plane:=1;          { Speicherebene C0 aktive    }
saveplane:=1;
init2;
help:=false;
helpbild;
{=====================Auswahl-Loop der Tastenfunktionen=====================}
repeat
     read(kbd,taste);
if not help
then begin
      case ord(taste) of

       72: helpbild;                     { H = Funktionstasten         }
      104: modus;                        { h = Modus anzeigen          }
       76: diskbild(true);               { L = laden Bildschirm        }
       83: diskbild(false);              { S = speichern Bildschirm    }
       77: movebild;                     { M = Bild horizontal/vertikal }
       67: scroll(false);                { C = Bildschirm scrollen     }
       80: printscr;                     { P = Bildspeicher drucken    }
      100: begin draw:=not draw; display;end;{ d = zeichnen ein/aus     }
      114: setrstproc;                   { r = Punkt setzen/loeschen   }
       70: findfadenkreuz;               { F = Fadenkreuz finden       }
       84: textproc;                     { T = Textmodus ein           }
       71: if (fontsw and (size < (vert div 14))) or
              (not fontsw and (size < (vert div 10)))
           then
              size:=size+1;              { G = Text vergroeaern        }
      103: if size > 1  then size:=size-1; { g = Text verkleinern      }
      111: clrbox:= not clrbox;          { o - Charakterbox loeschen   }
      102: fontsw:=not fontsw;           { f = font 8x8/8x14 ein/aus   }
      118: vertext:=not vertext;         { v = text vertikal schreiben }
      105: invertext:=not invertext;     { i = invertiert text schreiben }

       90: clearplane;                   { Z = Speicherebene loeschen  }
      120: begin xy:=not xy; displayxy; end; { x = x/y vom Fadenkreuz anzeige }
       27: begin read(kbd,ft);           { ESC                         }
             case ord(ft) of
              59: planeselect(1);        { F1 = Plane C0               }
              60: planeselect(2);        { F2 = Plane C1               }
              61: planeselect(3);        { F3 = Plane C0 und C1        }
              64: drawline;              { F6 = draw line              }
              89: xyline;                { shift F6 saves x/y-Position }
              65: copywindow(0);         { F7 = copy window 0          }
              66: copywindow(1);         { F8 = copy window 1          }
              67: copywindow(2);         { F9 = copy window 2          }
              68: copywindow(3);         { F10= copy window 3          }
              90: savex1y1(0);           { shift F7 = window 0         }
              91: savex1y1(1);           { shift F8 = window 1         }
              92: savex1y1(2);           { shift F9 = window 2         }
              93: savex1y1(3);           { shift F10= window 3         }
             100: savex2y2(0);           { ctrl  F7 = window 0         }
             101: savex2y2(1);           { ctrl  F8 = window 1         }
             102: savex2y2(2);           { ctrl  F9 = window 2         }
             103: savex2y2(3);           { ctrl  F10= window 3         }
              75: fadenkreuz(-1,0);      { nach links                  }
              77: fadenkreuz(1,0);       { nach rechts                 }
              72: fadenkreuz(0,-1);      { nach oben                   }
              80: fadenkreuz(0,1);       { nach unten                  }
             end;
           end;


       48: begin speed:=10;  display; end;   { 0 = 10 Punkte }
       49: begin speed:=1;   display; end;   { 1 =  1   "    }
       50: begin speed:=2;   display; end;   { 2 =  2.  "    }
       51: begin speed:=3;   display; end;
       52: begin speed:=4;   display; end;
       53: begin speed:=5;   display; end;
       54: begin speed:=6;   display; end;
       55: begin speed:=7;   display; end;
       56: begin speed:=8;   display; end;
       57: begin speed:=9;   display; end;   { 9 =  9 Punkte }


       42: begin speed:=speed*10; display; end; { speed * = 10 Punkte }
       43: begin speed:=speed+1;  display; end; { speed + 1 }
       45: begin speed:=speed-1;  display; end; { speed - 1 }
      end;
     end
else if taste = 'H'
     then
        helpbild;
until taste = 'E';                 { Taste "E" Programmende  }


Speicherebene(mode);
plane:=3;                          { C0 und C1 }
gotoxy(23,33);
write('Programm-Ende. Letzte Moeglichkeit !!');
diskbild(false);                   { letzte Moeglichkeit das Bild zu speichern }


   grafikEGA(false);               { Grafikmodus ausschalten }
   grafiktext(false);              { Text in Grafik aus }
end.
```

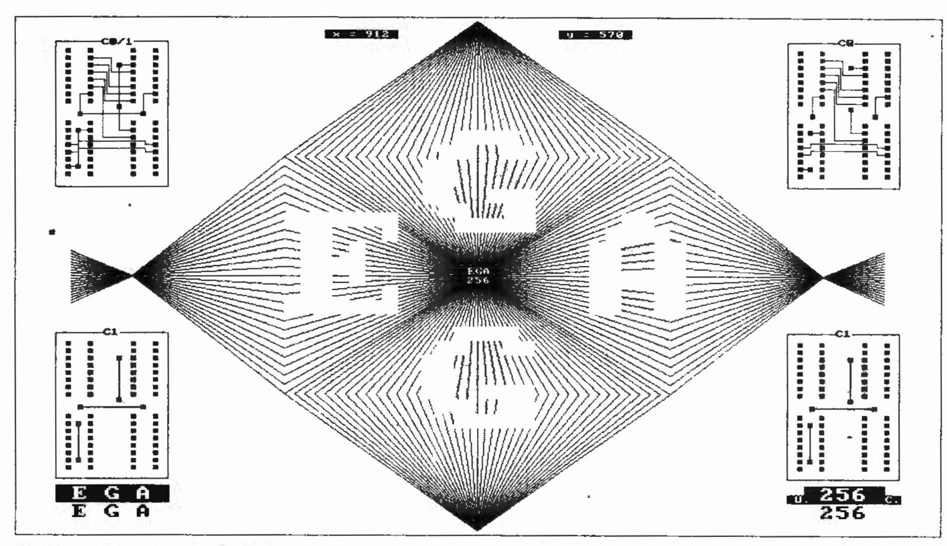


Bild 2. Ein Beispiel für die Anwendung von EGA-TCAD, das mit einer Auflösung von 912 × 570 Punkten erstellt wurde

der Bildschirmspeicher für den Bildrefresh ausgelesen wird.

256 KByte Bildspeicher

Und nun zum 256-KByte-Bildschirmspeicher. Bei der 256K-EGA sind alle vier Speicherebenen (C0, C1, C2 und C3) mit je 64 KByte ausgerüstet. Dadurch können alle 16 Palette-Register des Attribute Controllers adressiert werden (siehe Bild 2 in mc 4/87). Für den EGA-Modus 10H (Enhanced Color Bildschirm) sind die Palette-Register richtig initialisiert. Für den EGA-Modus 0FH (monochromer Bildschirm) sind jedoch nur die Palette-Register der Speicherebenen C0 und C2 sowie C3 für die „Blinking"-Funktion initialisiert.
Das Programmbeispiel zeigt, wie für den EGA-Modus 0FH die restlichen Palette-Register genutzt werden.
An dieser Stelle muß noch ein Bit im „Mode Control Register" des Attribute Controllers erklärt werden, das Bit 3 (Enable Blinking). Ist Bit 3 gesetzt, so wird eine Leitung, nennen wir sie blinking, periodisch ein- und ausgeschaltet. Diese Leitung ist – logisch gesehen – mit der „seriellen Datenleitung C3" XOR-verknüpft (Bild 2a). Die Leitung C3 ist eine der vier Adreßleitungen zu den Palette-Registern des Attribute Controllers. Ist zum Beispiel nun noch die Adreßleitung C2 aktiv, so wird periodisch im Attribute Controller das Palette-Register 0CH oder 04H adressiert (Bild 3 und 4 in mc 2/87).
Im Palette-Register 04H, EGA-Modus 0FH, ist der Farbwert 18H (intensified video) und im Palette-Register 0CH der Farbwert 00 (black) initialisiert. Das heißt, daß diese Bildschirmposition zwischen intensified video und schwarz wechselt. Durch entsprechende Initialisierung der Palette-Register kann zwischen beliebigen Farben (monochrom oder Farbe) „blinking" gesteuert werden, z. B. rot und blau. Wird das Bit 3 im „Mode Control Register" des Attribute Controllers auf 0 gesetzt, so ist „blinking" ausgeschaltet und die Speicherebene C3 steht voll als Bildspeicherebene zur Verfügung (normale Initialisierung bei EGA-Modus 10H).
Das Programm „EGA-TCAD" zeigt, welche Fähigkeiten in der 256K-EGA-Karte im Grafikmodus stecken. Das Programm basiert zum Teil auf dem bereits in mc 4/87 vorgestellten Programm, das jedoch jetzt mit allen vier Bildspeicherebenen arbeitet und sowohl im Grafik-Textmodus als auch im reinen Grafikmodus die Fähigkeiten der EGA voll nutzt.

Programmbeschreibung

Das Programm EGA-TCAD erlaubt die Arbeit mit beliebigen Bildgrößen, angefangen bei 640 × 350 über 640 × 818 bis zu 1488 × 352. Alle dazwischen liegenden Bildgrößen sind erlaubt, solange der Speicherbedarf 256 KByte nicht überschreitet.
Eine EGA-Karte mit 256 KByte ist also grundsätzlich erforderlich, sowie ein monochromer oder Enhanced Color Bildschirm. Über Tastaturfunktionen lassen sich beliebige Zeichnungen auf dem Bildschirm erstellen. Diese Zeichnungen können sowohl auf der Diskette gespeichert als auch mit dem Drucker ausgegeben werden. Zum Beispiel wurde mit diesem Programm bereits der Verdrahtungsplan von zweiseitigen Leiterplatten hergestellt. Eine Maus wurde hier aus gutem Grund nicht verwendet, was später an entsprechender Stelle erläutert wird.

Bildspeicher-Struktur

Die vier Bildspeicherebenen werden mit C0, C1, C2 und C3 bezeichnet. Für die Zeichnungen stehen zwei Speicherebenen zur Verfügung: C0 ist die Zeichenebene 0, C1 die Zeichenebene 1. Die Bildspeicherebene C2 wird als Fadenkreuz- und Hilfsebene eingesetzt, während C4 als Modus- und Fixpunktebene dient. Im Folgenden werden die einzelnen Programmfunktionen sowie die wichtigsten Pascal-Prozeduren erklärt.

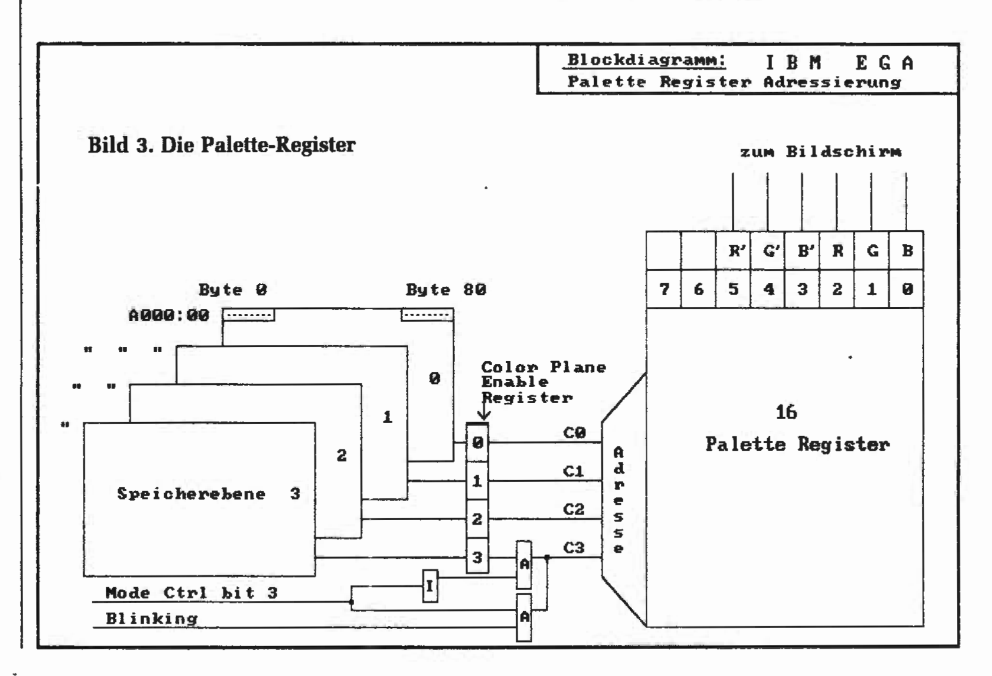


Bild 3. Die Palette-Register

Programm-Funktionen

Nach dem Programmstart „EGA-TCAD" muß man zuerst die gewünschte Bildgröße (z. B. Horizontal = 912 und Vertikal = 570) angeben. Die beiden x/y-Werte werden auf Gültigkeit geprüft. Die voreingestellten Standardwerte sind 640 × 350 Punkte. Das Produkt aus x * y darf 256 KByte (524 280 Punkte minus eine Scan-Zeile) nicht überschreiten. Ist der Wert von x kleiner als 640 oder der Wert von y kleiner als 350, so wird x mit 640 und y mit 350 angenommen.
Wurden beide Werte vom Programm akzeptiert, erscheint auf dem Bildschirm das „Tasten-Funktions-Bild" (*Bild* 4). In der ersten Zeile werden die gewählten Werte für die Horizontale x und die Vertikale y angezeigt. Die Horizontale x wird, wenn sie bei der Eingabe nicht berücksichtigt wurde, immer auf die nächste 16-Bit-Grenze (2 Byte) gesetzt. In den folgenden Zeilen wird eine Kurzbeschreibung der Tasten-Funktionen angezeigt. Außer bei den Funktionen „C", „M" und „T" kann das Tasten-Funktions-Bild immer aufgerufen werden.

```
           x=912          E G A - T C A D        y=570
---Taste----------Funktion----------------------------------------------
  H/h =>          Tastenfunktionen / Modus anzeigen, ein/aus.
  E  =>           Programmende.
  L  =>           Bildschirmspeicher von der Diskette laden.
  S  =>           Bildschirmspeicher auf die Diskette speichern.
  M  =>           Bild auf dem Bildschirm rotieren, ein/aus. (Cursor Tasten).
  C  =>           Bildschirm scrollen, oben/unten/links/rechts. (Cursor Tasten)
  P  =>           Bildschirm auf dem Grafikdrucker drucken. (ITHO 8510 SP).
  d  =>           zeichnen, ein/aus.
  r  =>           Punkt setzen = "S", Punkt löschen = "R", (ein/aus).
 cursor  =>       Fadenkreuz nach links/rechts/oben/unten bewegen.
  F,x   =>        Fadenkreuzes finden; x/y Position anzeigen, ein/aus.
1..9,0,*,+,- =>   Bewegungszahl des Fadenkreuzes, Punkte/cursor Taste.
  F1 =>           Zeichenebene C0.
  F2 =>           Zeichenebene C1.
  F3 =>           Zeichenebene C0 und C1.
  F6 =>           Linie von x/y (shift F6) nach x/y von Fadenkreuz ziehen.
  Z  =>           Zeichenebene Cx löschen.
 F7..F10 =>       Selektion/Kopie Fenster 0..3 (shift/ctrl).
  T  =>           Textmodus einschalten. Textende = Eingabetaste.
  G  =>           Scalefaktor für Text um 1 erhöhen.
  g  =>           Scalefaktor für Text um 1 erniedrigen.
  o  =>           Charakterbox im Textmodus löschen.
  f  =>           Fontumschaltung von 8x8 nach 8x14 und zurück.
  v  =>           Text vertikal schreiben, ein/aus.
  i  =>           Text invertiert schreiben, ein/aus.
------------------------------------------------------------------------

         zurück =>    Taste "H".
```

Bild 4. So sind die Tasten belegt

Anmerkungen zu Bild 3

Ist durch die Funktion „C" (scrollen) der Modus aus dem Bild gewandert, so kann er mit „h" wieder zurückgeholt werden (ein/aus). Nach der Wahl von „E" (Programmende) wird zur Sicherheit gefragt, ob der Bildspeicher auf die Diskette gespeichert werden soll. Hat man beim Ladebefehl „L" keinen Dateinamen angegeben, wird die Funktion ignoriert. Beim Speichern der selektierten Ebene(n) werden, sofern das Bild weniger als 32 KByte belegt, 32 KByte auf die Diskette geschrieben. Benötigt das Bild mehr als 32 KByte, werden 64 KByte auf die Diskette geschrieben. Ohne Angabe eines Dateinamens wird die Funktion ignoriert.
Der Inhalt der selektierten Bildspeicherebenen wird mit den Cursortasten nach links/rechts, oben/unten rotiert, wenn man die Funktion „M" eingeschaltet hat. Nach Auswahl der Funktion „C" wird der Bildschirminhalt nach links/rechts, oben/unten gescrollt.
Wird mit „d" das Zeichnen ausgeschaltet, kann das Fadenkreuz frei bewegt werden. Nach Eingabe von „x" wird die Position des Fadenkreuzes angezeigt. Nach Eingabe von „F" wird das Fadenkreuz in der linken oberen Ecke des Bildschirmes positioniert, unabhängig von der Position des Bildschirmes im Bildschirmspeicher. Mit „Z" kann man die selektierte Bildspeicherebene(n) löschen. Die Tastenkombination „Shift-F6" setzt den Startpunkt für das Zeichnen einer Linie, die mit „F6" bis zum Fadenkreuz gezeichnet wird.
Mit den Funktionstasten F7...F10 kann ein Fenster von „a" nach „b" kopiert werden. Zuvor muß z. B. mit „Shift-F7" die linke obere Ecke und mit „CTRL-F7" die rechte untere Ecke des zu kopierenden Fensters markiert werden. An den markierten Punkten bleibt ein Fadenkreuz als Anzeige zurück. Mit dem weiter beweglichen Fadenkreuz wird nun die linke obere Ecke auf dem Bildschirm angefahren, wohin das Fenster kopiert werden soll. Die Funktion „r" muß auf „S" (Punkt setzen) stehen. Steht die Funktion „r" auf „R" (Punkt löschen), so wird ein angefahrenes Feld von der Größe des Fenster gelöscht. Mit den vier Tasten F7...F10 können vier verschiedene Fenster gleichzeitig gespeichert werden.
Der Textmodus wird mit „T" eingeschaltet. Die Position des Fadenkreuzes markiert den Startpunkt des Textes. Der Text kann an jeder beliebigen x/y-Position beginnen. Der Textmodus wird mit der Taste „Eingabe" beendet. Zur Auswahl der Charaktergröße dient die Funktion „G". Normalerweise ist die Charakterbox 8 × 10 oder 8 × 14 Punkte groß und G hat den Wert 1. Wird die Taste „G" gedrückt, so wird der Wert von G um 1 erhöht. Ist „G" größer als 1, so wird die Charakterbox G × 8/G × 10 oder G × 8/G × 14 groß. „G" ist also der Scalefaktor der Charakterbox. Wird „g" gedrückt, so wird der Scalefaktor G um den Wert 1 erniedrigt. Kleiner als 1 kann der Wert von G nicht werden.
Mit „o" wird entschieden, ob die Textbox vor der Eingabe eines Textes gelöscht werden soll. Wird die Textbox nicht gelöscht, kann der Text über eine schon vorhandene Zeichnung geschrieben werden, ohne daß die in der Textbox befindliche Zeichnung gelöscht wird. Der Text wird bei vertikaler Schrift vom Fadenkreuz aus senkrecht nach unten geschrieben. Die einzelnen Textfunktionen werden erst in der Programmstatuszeile sichtbar, wenn die Funktion Text „T" aktiviert wird. Will man sie ändern, muß der Textmodus durch „Eingabe" verlassen werden. Nachdem die entsprechende Textfunktion gewählt wurde, kann mit „T" an der gleichen Stelle weitergemacht werden. Ist die Charakterbox für die Startposition im Bild zu groß, so wird der Charakter nicht geschrieben.

Bilddaten auf der Diskette

Jedes Bild, das auf die Diskette gespeichert wird, besteht pro Bildspeicherebene aus zwei Datensätzen. Zum einen aus der Bildspeicherebene C0, den Dateien „NAME.C0" (Bild x/y VAR C0), „NAME.C0V" und/oder der Bildspeicherebene C1, mit den Dateien „NAME.C1" (Bild x/y VAR C1), „NAME.C1V".
In der Datei der Bildspeicherebene befindet sich der Bildinhalt, während in der Datei (Bild x/y Var) nur die Höhe und Breite des Bildes gespeichert ist. Um aus dem Bild der „Tastenfunktionen" herauszukommen, betätigt man die

Taste „H“. Jetzt wird der ganze Bildschirm gelöscht und in der letzten Zeile wird der Programmstatus angezeigt.

Programmstatus

Die Anzeige „Plane: C0“ besagt, daß die Speicherebene C0 als Zeichenebene selektiert ist. Sind z. B. C0 und C1 selektiert, so wird gleichzeitig in beiden Speicherebenen gezeichnet.
Der Modus zeigt an, in welchem Arbeitsmodus sich das Programm gerade befindet z. B.:
bewegen: Bewegung des Fadenkreuzes ohne zu zeichnen.
scrollen: Bild nach oben, unten, links, rechts bewegen.
zeichnen: Abhängig von „S“ oder „R“ wird bei der Bewegung des Fadenkreuzes entweder gezeichnet oder gelöscht.
Bei Disketten-Operationen muß ganz rechts in der Programmstatuszeile noch der Filename angegeben werden.

Fadenkreuz-Bewegung

Die Bewegung des Fadenkreuzes kann mit den Tasten 1...9, 0, *, +, – sowie den Cursortasten gesteuert werden. Hiermit kann man Bewegungszahlen von 1 bis 256 einstellen. Sollte das Fadenkreuz einmal nicht zu sehen sein, z. B. nach der Funktion „scrollen“, so kann es mit der Funktion „F“ in die obere linke Ecke des Bildschirmes transportiert werden. Dabei ist es egal, wo die Anfangsadresse des Bildschirms sich gerade im Bildschirmspeicher befindet.

Ein praktisches Beispiel

Um die Lötaugen eines IC-Sockels zu zeichnen wird mit „4“ ein Quadrat von 5 × 5 Punkten gezeichnet, und in dieses hinein noch ein Quadrat von 3 × 3 Punkten. Dieses Lötauge wird nun auf die Taste F7 gespeichert. Die Anzahl der Punkte wird auf „8“ gesetzt. Mit den beiden Tasten z. B. „Cursor nach rechts“ und „F7“ können nun an einer beliebigen Stelle für einen IC-Sockel acht Lötaugen kopiert werden. Diese acht Lötaugen kann man nun unter Taste F8 speichern und dann in entsprechendem Abstand unter die acht Lötaugen kopieren. Dieses Beispiel zeigt deutlich, daß in diesem Fall die Cursortasten einer Maus überlegen sind.

Die wichtigsten Prozeduren

Hier werden nur die Prozeduren beschrieben, die neu sind oder sich gegenüber denen in mc 4/87 bereits beschriebenen wesentlich geändert haben.
Alle Prozeduren, so weit betroffen, sind auf die 256K-EGA abgestimmt. Die Datei „Grafik4.EGA“ (*Bild 5*) ist eine Sammlung von Prozeduren, die die EGA direkt ansteuern. Für andere Anwendungen kann diese Datei entsprechend erweitert oder gekürzt werden. Bis auf die Grafiktext-Prozeduren „Gotoxy, writechar und writetext“ sind alle Prozeduren unabhängig geschrieben.

Funktion pd(x,y,farbe,maske)

Diese Funktion testet, ob an der Position x,y ein Punkt mit der Farbe „farbe“ steht. Hier wird zum ersten Mal die Color-Compare-Funktion des Grafikcontrollers benutzt. Hat die Variable „maske“ den Wert 0, so werden nur die Speicherebenen getestet, die durch die Variable „farbe“ gegeben sind. Wird z. B. die Variable „maske“ auf 0FH gesetzt, wird der Test auf allen vier Speicherebenen durchgeführt. Ist der Test erfolgreich, wird in „pd“ die Variable „farbe“, ansonsten der Wert 0 zurückgegeben.

Prozedur writechar(c), writetext(c)

Diese Prozeduren haben sich gegenüber dem Programm in mc 4/87 wesentlich geändert. Der Text kann jetzt auch an jeder beliebigen horizontalen Position beginnen. Ebenso kann der Text beliebig vergrößert/verkleinert werden.

Prozedur grafiktext(ein)

Diese Prozedur schaltet den Grafiktext-Modus ein/aus. Sie ist nur nötig, wenn

```
{*****************************************************************************}
{*                            I B M   E G A                                  *}
{*                  Turbo Pascal Grafiks-Unterstuetzung                      *}
{*   Ulrich Cebulla                                                          *}
{*   7036 Schoenaich, Hermann Werner Str.6                  21.05.87         *}
{*****************************************************************************}

{* Filename: Grafik.EGA *}
const
    egabase=$A000;                  { Startadresse des Bildschirmspeichers  }
var
   EGAmode,savemode         :byte;    { 0F=Monochrome, 10=Enhanced Color Disp.}
   vert                     :integer; { vertikale Bildhoehe                   }
   vscrollpos               :integer; { vertikale Srollposition               }
   hor                      :integer; { horizontale Bildbreite in bytes       }
   horoffset                :integer; { CRT-Offset in words (2 Bytes)         }
   hscrollpos               :integer; { horizontale Scrollposition            }
   horpel                   :integer; { pels/scan line                        }
   startzeile               :integer; { Bildstart-Adresse                     }
   scrolline                :integer; { Scan line Laenge                      }
   xtext,ytext              :integer; { Text x/y Position                     }
   lastxtext,lastytext      :integer; { letzte x/y Position                   }
   nextxy                   :boolean; { true, wenn noch kein Zeilenende       }
   wrtinvert,wrthor,graftext:boolean; { Text Schreibkontrolle                 }
   fontsegm,fontoffs        :integer; { Font pointer                          }
   conoutptrsave            :integer; { Turbo Pascal "ConOutPtr"              }
   pattern,box              :byte;    { bytes/charakter                       }
   size                     :byte;    { Vergroesserungsfaktor fuer Charakter  }
   clrbox                   :boolean; { Charakterbox loeschen                 }
   i,j                      :integer; { lauf var                              }
   a                        :byte;    { dummy var                             }
{======================Grafik Text Write/ln Proceduren========================}
```

```
{---------------------Font Pointer-------------------------------------------}
procedure Fontpointer(font:boolean);      { Diese Procedure ermittelt die     }
                                          { fontsegment und fontoffset Adresse}
                                          { von Font 8x8 und Font 8x14 im EGA }
                                          { ROM. }
type regs=record case integer of
            1:(ax,bx,cx,dx,bp,si,di,ds,es,flgs: integer);
            2:(al,ah,bl,bh,cl,ch,dl,dh: byte);
          end;
  var reg:regs;
begin
        reg.ax:=$1130;         { AH = EGA information              }
        if font
        then begin
             reg.bh:=3;        { 8x8 font }
             pattern:=8;       { 8 Bytes/charakter }
             box:=10;          { Box-Groesse = 10 }
             end
        else begin
             reg.bh:=2;        { 8x14 font }
             pattern:=14;      { 14 Byte/Charakter }
             box:=14;          { Box-Groesse = 14 }
             end;

        intr($10,reg);
        fontsegm:=reg.es;      { Font Segment-Adresse }
        fontoffs:=reg.bp;      { Font Offset-Adresse  }

end;

{---------------------Pascal Punkt setzen------------------------------------}
{ Punkt setzen an der Position x/y, Farbe = color, Funktion = funktion       }
```

Bild 5. In der Datei Grafik4.EGA sind zahlreiche nützliche Funktionen zusammengefasst

```
procedure pset(x,y:integer;color,funktion:byte);
var offset :integer;
    bitmask:byte;
begin

bitmask   := $80 shr (x and 7 );  { x-Bit Position im Byte }
offset    := (y * hor)+(x shr 3);

                                 { GK = Grafiks-Kontroler }

      port[$3ce] := 0;           { GK SET/RESET Reg. }
      port[$3cf] := color;       { Farbe ins SET/REST Reg. }
      port[$3ce] := 1;           { GK ENABLE SET/RESET Reg.}
      port[$3cf] := $0F;         { erlaube alle Planes }
{ Folgende 2 Instruktionen sind nur noetig,wenn die "Funktion" benutzt wird }
{     port[$3ce] := 3;  }        { GK Rotate Reg. }
{     port[$3cf] := (funktion and $03) shl 3;}
                                 { Funktion  00 unmodifiziert          }
                                 {           01 UND mit gelesenen Daten }
                                 {           10 OR  "        "     "  }
                                 {           11 XOR "        "     "  }

      port[$3ce] := 8;           { GK BIT MASK Reg. }
      port[$3cf] := bitmask;     { PSET nur das durch x/y ausgewaehlte bit }

      mem[egabase:offset]:=mem[egabase:offset]; { read/write, siehe Text     }
                                                { ueber den Grafik Controller}

      port[$3cf] := $FF;         { erlaube wieder alle bits in GK Bi. Mask Reg.}

      port[$3ce] := 1;           { GK ENABLE SET/RESET Reg. }
      port[$3cf] := 0;           { disable alle Planes }
{ Folgende 2 Instruktionen sind nur noetig,wenn die "funktion" benutzt wird }
{     port[$3ce] := 3; }         { GK Rotate Reg. }
{     port[$3cf] := 0; }         { Funktion 00    }
end;
{------------------------Punkt testen----------------------------------------}
function pd(x,y:integer;farbe,maske:byte):byte;
var bitmask,p:byte;
    offset   :integer;
begin
  bitmask   := $80 shr (x and 7 );  { x-Bit Position im Byte }
  offset    := (y * hor)+(x shr 3);

  port[$3ce]:=2;               {color compare Register}
  port[$3cf]:=farbe;
  port[$3ce]:=7;               {color don't care Register}
  port[$3cf]:=farbe or maske;
  port[$3ce]:=5;               {Mode Register}
  port[$3cf]:=$08;             {read mode 1}
  p:=mem[egabase:offset] and bitmask;
  port[$3cf]:=0;
  if p <> 0
  then
      pd:=farbe
  else
      pd:=0;
end;

{-----------------------Turbo Pascal Gotoxy--------------------------------}
procedure turbogotoxy(x,y:integer);
begin
  Gotoxy(x,y);                 { Turbo Pascal Gotoxy }
end;

{--------------------------Grafik Gotoxy------------------------------------}
procedure GotoXY(X,Y:integer);  { Diese Procedur GotoXY ersetzt die standart  }
                                { Procedur im Turbo pascal, wenn der EGAmode  }
                                { $0F oder $10 ist. Ist EGAmode 0, so wird    }
begin                           { ueber die Procedur "turbogotoxy" die Turbo- }
                                { Pascal Gotoxy aufgerufen.                   }

  if EGAmode = 0 then turbogotoxy(x,y); { => Turbo Gotoxy }
  xtext:=X;
  ytext:=Y;
end;

{---------------------------Write Text im Grafikmodus-----------------------}
procedure writechar(c :byte);     { Diese Procedure schreibt den Charakter }
var offset,i,i0,i1,i2,i3,i4,xtt,ytt : integer;   { "c" an die Position x/y.      }
    inv,patt,mask,p1,p2 : byte;
begin
  if graftext
then begin
     ytt:=ytext;
     xtt:=xtext;

     if wrtinvert
     then
         inv:=$0f                     { invertiert alle charakter }
     else
         inv:=0;

     if (box = 10) and clrbox       { Boxgroesse 10 bei 8x8 Font }
     then begin
          for i:=1 to size
          do begin
             i4:=0;
             for i1 := 1 to 8*size
             do begin
                pset(xtt+i4,ytt,$00 xor inv,0);
                i4:=i4+1;
                end;
             ytt:= ytt+1;
             end;
          end;

     for i := 0 to pattern-1        { Charakter Font loop }
     do begin
        patt:=mem[fontsegm:fontoffs+(c*(pattern))+i];
        for i0:= 1 to size
        do begin
           mask:=$80;
           i4:=0;
           for i2:= 0 to 7
           do begin
              for i3:=1 to size
              do begin
                 if patt and mask = 0
                 then begin
                      if clrbox
                      then
                          pset(xtt+i4,ytt,$00 xor inv,0)
                      end
                 else
                     pset(xtt+i4,ytt,$0f xor inv,0);
                     i4:=i4+1;
                 end;
              mask:=mask shr 1;
              end;
           ytt:=ytt+1;       { naechste scan line }
           end;
        end;

     if (box = 10) and clrbox        { Boxgroesse 10 bei 8x8 Font }
     then begin
          for i:=1 to size
          do begin
             i4:=0;
             for i1 := 1 to 8*size
             do begin
                pset(xtt+i4,ytt,$00 xor inv,0);
                i4:=i4+1;
                end;
             ytt:= ytt+1;
             end;
          end;
     end
else begin
     offset := xtext - 1 + ((ytext - 1) * (hor*box));   { im Bildspeicher }

     if wrtinvert
     then
         inv:=$ff                     { invertiert alle charakter }
     else
         inv:=0;

     if box = 10                        { Boxgroesse 10 bei 8x8 Font }
     then begin
          mem[egabase:offset]:= inv;   { + 1 blank/invert Kopfbyte/Charakter}
          offset:= offset+hor;
          end;

     for i := 0 to pattern-1         { Charakter Font loop }
     do begin
```

```
        mem[egabase:offset]:=mem[fontsega:fontoffs+(c*(pattern))+i] xor inv;

        offset:= offset + hor;        { naechste scan line }
        end;

      if box = 10                       { Boxgroesse 10 bei 8x8 Font }
      then
          mem[egabase:offset]:= inv;    { + 1 blank/invert Fussbyte/Charakter }
      end;
end;

{---------------Start-Procedur fuer Turbo Pascal "write/ln"-------------------}
procedure writetext(c: byte);      { Diese Procedure ist die Startprocedure   }
                                   { von write(ln), im Grafik-Modus. Die x/y-  }
                                   { Position, und die Kontrollcodes BS,LF und }
                                   { CR werden verarbeitet.                   }
                                   { Kein automatischer LF weder horizontal   }
                                   { noch vertikal.                           }
begin

  lastxtext:=xtext;               { letzte x-Position      }
  lastytext:=ytext;               { letzte y-Position      }
  case c of
{---------------BS (Backspace)--------------}
     8:   if wrthor
          then begin
               if graftext
               then begin
                    if xtext > (size*8)
                    then xtext:=xtext-(size*8);
                    end
               else begin
                    if xtext > 1
                    then
                         xtext:=xtext - 1;            { <= 1 Charakter }
                    end;
               end
          else if graftext
               then begin
                    if ytext >= box*size
                    then
                        ytext:=ytext - (box*size);   { 1 Char.-Box nach oben }
                    end
               else if ytext > 1
                    then
                        ytext:=ytext - 1;         { 1 Charakter nach oben }
{---------------LF (Line feed)--------------}
    10: if not graftext
        then begin
             if (box = 10) and (ytext < 35) or
                (box = 14) and (ytext < 25)
             then
                 ytext:=ytext + 1;              { naechste Zeile }
             end;
{---------------CR (Carriage return)--------}
    13: xtext:= 1;                              { Spalte 1 }
{------------------alle Charakter------------}
  $00..$FF: begin
            if (graftext and
               (xtext <= hor*8-(8*size)) and
               (ytext <= vert-(box*size))) or
               not graftext
            then
            {-----------}
            writechar(c);                         { Charakter schreiben }
            {-----------}

            if wrthor                           { Text horizontal       }
            then begin
                 if graftext
                 then begin
                      if xtext <= hor*8-(8*size)
                      then xtext:=xtext+(8*size);
                      end
                 else
                      if xtext < hor
                      then
                           xtext:=xtext + 1;    { naechste x-Position (char)}

                 end
            else if graftext                    { Text vertikal,graftext }
                 then begin
                      if ytext <= (vert-(2*box*size)+1) { nicht ueber "vert" }
                      then
                          ytext:=ytext + (box*size);   { naechste y-Position }

                      end
                 else if (box = 10) and (ytext < 35) or
                         (box = 14) and (ytext < 25)
                      then                            { not graftext }
                          ytext:=ytext + 1;           { naechste y-Position }
            end;
  end; { case of c }

  if (lastxtext <> xtext) or (lastytext <> ytext)
  then
      nextxy:=true                          { neue x/y-Position }
  else
      nextxy:=false;                        { Zeilenende erreicht }
end;

{#################### I B M   E G A  Grafik-Hilfs-Proceduren################}

{---------------------Grafik Modus EIN/AUS schalten------------------------}
procedure grafikEGA(on:boolean);

type regs=record case integer of
          1:(ax,bx,cx,dx,bp,si,di,ds,es,flgs: integer);
          2:(al,ah,bl,bh,cl,ch,dl,dh: byte);
          end;
  var reg:regs;
begin
    if on                       { Grafiksmode einschalten }
    then begin
         reg.ax:=$0F00;         { AH = 0F lesen des EGA Status }
         intr($10,reg);
         savemode:=reg.al;      { EGA mode in AL }

         reg.ax:=$1200;         { Alternate select }
         reg.bl:=$10;           { EGA-Information }
         intr($10,reg);

         EGAmode:=0;
         if reg.cl in [2,3,8,9]   { Enhanced Color Display ? }
         then
             EGAmode:=$10;

         if reg.cl in [4,5,10,11]  { Monochrome Display }
         then
             EGAmode:=$0F;

         if (EGAmode = 0) or (reg.bl <> 3)
         then begin
              writeln('EGA Grafik 640x350 mit dieser Karte nicht moeglich');
              if reg.bl <> 3
              then
                  writeln('EGA Speicher < 256k Bytes');
              writeln(reg.bl,' ',reg.cl);
              Halt;       { Programm HALT !!!! }
              end;

         reg.ah:=$00;                { EGA Modus }
         reg.al:=EGAmode;            { EGAmode $0F/$10 setzen }
         intr($10,reg);              { EGA BIOS Intrrupt $10 }
         scrolline:=hor;             { 1 scan = 80 Bytes }
         end
    else begin                       { Grafiks Mode ausschalten }
         EGAmode:=0;
         reg.ah:=0;
         reg.al:=savemode;           { zurueck zu letztem EGA Modus }
         intr($10,reg);              { EGA BIOS Interrupt $10 }
         end;
end;
{---------------------Textmodus im Grafikmodus Ein/Aus schalten-------------}
Procedure grafiktext(ein:boolean);
begin
   if ein
   then begin
        conoutptrsave := conoutptr;  { Turbo Pascal conoutptr speichern }
        conoutptr := ofs(writetext); { Grafik write/ln interface }
        wrtinvert := false;          { normal/reverse video (normal) }
        wrthor    := true;           { horizontal/vertikal schreiben (horiz.)}
        graftext:=true;              { erlaube jede scan line (y-Achse)}
        size:=1;                     { Vergroesserung = 1 }
        clrbox:=true;                { Charakterbox loeschen }
```

```
        fontpointer(true);          { true = 8x8 font im grafik text }
        end
     else
        conoutptr:=conoutptrsave;   { zurueck zu Turbo Pascal "write/ln"   }
end;
{--------------------Bildschirm invertiern--------------------------------}

procedure Invertbild;       { 640x350 Bildschirm invertieren }
  begin
    port[$3ce]:=0;                  { GK Register 0               }
    port[$3cf]:=15;                 { alle bits set/reset         }
    port[$3ce]:=1;                  { GK Register 1               }
    port[$3cf]:=15;                 { erlaube alle bits           }
    port[$3ce]:=3;                  { GK Register 3               }
    port[$3cf]:=$18;                { XOR Funktion                }
                                    { invertiere alle Bytes des ganzen }
    move(mem[egabase:0],mem[egabase:0],28000);
                                    { sichtbaren Bildschirmspeichers  }
    port[$3cf]:=0;                  { Funktion 00                 }
    port[$3ce]:=1;                  { GK Register 1               }
    port[$3cf]:=0;                  { disable set/reset           }

  end;

{--------------------Farbpalettenregister setzen-------------------------}
procedure Farbpalette(farbe,palettereg:integer);
  var regs:record case integer of
          1:(ax,bx,cx,dx,bp,si,di,ds,es,flgs: integer);
          2:(al,ah,bl,bh,cl,ch,dl,dh: byte);
        end;
begin
  with regs do
    begin                    { Farbpaletten Register setzen }
      ax:=$1000;
      bl:=palettereg and $0F;
      bh:=farbe      and $3F;
      intr($10,regs);
    end;
end;

{--------------------Schreibmaske fuer Speicherebene/n--------------------}
{ Note: Nur in die hier selektierte/n Bildspeicherebene/n des Bildschirmspei- }
{       chers kann geschrieben werden.                                        }

procedure Speicherebene(ebene:byte);
begin
     port[$3c4]:=2;      { Sequencer Adress Register= Map Mask Reg }
     port[$3c5]:=ebene;  { Map Mask Register, Schreibmaske fuer Speicherebene/n}
end;
{-------------------------------------------------------------------------}
Procedure EGAwritemode(mode:byte);  { EGA write mode, siehe Mode Register des }
                                    { Grafik-Kontrollers }
begin
     port[$3ce]:=5;      { Mode Register, Grafik-Kontroller }
     port[$3cf]:=mode;
end;
{--------------------Bildebene/n-Auswahl fuer Bildschirm-------------------}
{ Note: Nur die hier ausgewaehlte/n BildSpeicherebene/n werden auf dem Bild-  }
{       schirm angezeigt.                                                     }

procedure Bildebene(ebene:byte);
var a:byte;
begin
     if EGAmode = $0F
     then
          a := port[$3ba]  { Adressmodus, Attribute Kontroller (Monochrome)  }
     else
          a := port[$3da];  {    "           "          "      (Color      )  }

     port[$3c0] := $12;          { Color Plane Enable Register, Display AUS)
     port[$3c0] := ebene;        { Display Speicherebene/n                  }
     port[$3c0] := $20;          { Display EIN                              }
End;
{-------------------------------------------------------------------------}
Procedure speicherebeneread(ebene:byte);  { Nur die hier ausgewaehlte Bild-    }
                                          { speicherebene kann gelesen werden.}
begin
    port[$3ce] := 4;            { GK Read Map Select Reg. }
    port[$3cf] := ebene;
end;

{------------------Bildschirm CRT-Offset-Register setzen--------------------}
Procedure setcrtoffset;
begin
    if EGAmode = $0F
    then begin
         port[$3b4]:=$13;
         port[$3b5]:=lo(horoffset);
         end
    else begin
         port[$3d4]:=$13;
         port[$3d5]:=lo(horoffset);
         end;

    hor:=horoffset*2;
    scrolline:=hor;
    horpel:=horoffset*16-1;
    hscrollpos:=0;
    vscrollpos:=0;
end;
{------------------Bildschirm CRT Start Addresse setzen---------------------}
Procedure CRTstartaddr(startaddr:integer);
var
   addr:integer;
   x   :byte;
begin
if EGAmode = $0F
then
    addr:=$3ba        { Input Status Register (Monochrome) }
else
    addr:=$3da;       { Input Status Register (Color     ) }

repeat                       { lese "Input Status Register" so lange, bis das }
     x:=port[addr] and $08   { Signal "Vertical Retrace" (Bildsychronisation) }
until x = $08;               { true ist. Siehe Beshreibung der Procedur.      }

{ Delay(1); }                { Siehe Beschreibung }

if EGAmode = $0F           { CRT Start Address Register setzen }
then begin                      { Monochrome-Modus }
     port[$3b4] := $0c;         { CRT Kontroller Register $0C }
     port[$3b5] := hi(startaddr); { Start Adresse high }
     port[$3b4] := $0d;         { CRT Kontroller Register $0D }
     port[$3b5] := lo(startaddr); { Start Adresse low  }
     end
else begin                      { Color-Modus }
     port[$3d4] := $0c;         { CRT Kontroller Register $0C }
     port[$3d5] := hi(startaddr); { Start Adresse high }
     port[$3d4] := $0d;         { CRT Kontroller Register $0D }
     port[$3d5] := lo(startaddr); { Start Adresse low  }
     end;
end;
{------------------Blinken ein/ausschalten----------------------------------}
Procedure blinking(einaus:boolean);
begin
   if einaus
   then begin
        if EGAmode = $0F
        then begin
             a:=port[$3ba];
             port[$3c0]:=$10;
             port[$3c0]:=$08;    { Monochrome blinking ein }
             port[$3c0]:=$20;
             end
        else begin
             a:=port[$3da];
             port[$3c0]:=$10;
             port[$3c0]:=$09;    { Farbe blinking ein }
             port[$3c0]:=$20;
             end;
        end
   else begin
        if EGAmode = $0F
        then begin
             a:=port[$3ba];
             port[$3c0]:=$10;
             port[$3c0]:=$03;    { Monochrome blinking aus }
             port[$3c0]:=$20;
             end
        else begin
             a:=port[$3da];
             port[$3c0]:=$10;
             port[$3c0]:=$01;    { Farbe blinking aus }
             port[$3c0]:=$20;
             end;
        end;
end;
```

die Grafiktext-Prozeduren GotoXY, writechar und writetext in der Include-Datei GRAFIK4.EGA benutzt werden. Diese Prozedur muß dann direkt hinter der Procedur „grafikEGA“ aufgerufen werden.

Prozedur setcrtoffset

Mit dieser Prozedur wird die horizontale Länge einer Scan-Zeile in den CRT-Controller (CRT Offset Register) gesetzt.

Prozedur CRTstartaddr(startaddr)

Diese Prozedur ist ein Teil der ehemaligen Prozedur scroll. Sie enthält jetzt nur noch den Teil der Scroll-Funktion, der direkt mit den CRT-Registern kommuniziert.

Prozedur blinking(ein/aus)

Mit dieser Prozedur wird der Blinkmodus der EGA ein- und ausgeschaltet. Das gilt sowohl für den Monochrom- als auch für den Enhanced Color -Bildschirm. Hier muß allerdings berücksichtigt werden, daß die Palette-Register vorher entsprechend gesetzt sind.

Prozeduren im Programm EGA_TCAC (Bild 1): Prozedur egamem

Die Prozedur „egamem“ stellt sicher, daß die Eingabeparameter x/y weder nach unten (Bild kleiner als 640 × 350 Punkte) noch nach oben (Bild größer als 256 KByte) sind.

Prozedur findfadenkreuz

Durch diese Prozedur kann das Fadenkreuz mit der Variablen „startzeile“ immer in die linke obere Ecke des Bildschirms gebracht werden.

Prozedur planeselect(select)

Diese Prozedur wird durch die Tasten F1, F2 und F3 angesprungen. Hier wird die Auswahl der Speicherebene(n) getroffen, in die gezeichnet werden soll.

Prozedur modus

Durch diese Prozedur kann der Modus wieder am unteren Bildrand sichtbar gemacht werden. Das ist z. B. notwendig, wenn durch die Funktion „scrollen“ der Modus aus dem Bild gewandert ist und man z. B. wissen will ob „zeichnen“ ein oder aus ist. Mit „h“ kann man zwischen der Modus- und der aktiven Bildposition hin und her schalten.

Prozedur savescreen (filename,filemode)

Diese Prozedur schreibt als erstes den selektierten Bildspeicher auf die Diskette. Abhängig von der selektierten Speicherebene ist der Filemode entweder C0 oder C1. Die Größe des Datensatzes ist abhängig von der Bildgröße (32 KByte oder 64 KByte). Es muß also darauf geachtet werden, daß immer genügend Speicherplatz auf der Diskette zur Verfügung steht; maximal können es 128 KByte sein (C0 und C1). Als zweites werden die Vertikale (vert) und die Horizontale (horoffset) des Bildes als Datensatz mit dem Filemode C0V oder C1V auf die Diskette geschrieben.

Prozedur loadscreen (filename,filemode)

Diese Prozedur liest ein auf der Diskette gespeichertes Bild und lädt es in den Bildspeicher der EGA. Abhängig davon, welche Speicherebene(n) zur Zeit selektiert ist, werden die entsprechenden Bilder von der Diskette gelesen. Als erstes wird der entsprechende Datensatz mit dem Filemode C0/1V in die Variable „vert“ und die Variable „horoffset“ gelesen. Diese beiden Variablen werden wieder zur Initialisierung des CRT-Controllers (horoffset) und für die Bildvertikale benötigt. Es ist dabei unwichtig, welche Bildgröße zur Zeit des Einlesens spezifiziert war. Als nächstes wird das Bild in die Bildspeicherebene(n) geladen. Während des Einlesens ist der Bildschirm dunkel gesteuert.

Prozedur scroll(setstart)

Diese Prozedur enthält nur noch den logischen Teil der eigentlichen Scroll-Funktion. Diese Prozedur erlaubt vertikales und jetzt auch horizontales Scrollen. Horizontal werden mit einer Cursortaste immer acht Punkte (1 Byte) bewegt. Das liegt daran, daß das Start-Register des CRT-Controllers nur byteweise den Bildspeicher adressiert. Die innere Procedur CRTstartaddr übernimmt dann die Steuerung des CRT-Controllers.
Mit dem Attribute Register „Horizontal Pel Panning“ 3C0H.13H ist es selbstverständlich auch möglich horizontal Punkt für Punkt zu scrollen, wurde hier aber aus gutem Grund nicht gemacht.

Prozedur savex1y1(window)

Mit dieser Prozedur wird der linke obere Punkt des Fensters „window“ festgelegt und als Markierung ein Fadenkreuz in die Speicherebene „mode“ gesetzt.

Prozedur savex2y2(window)

Mit dieser Prozedur wird der rechte untere Punkt des Fensters „window“ festgelegt und als Markierung ein Fadenkreuz in die Speicherebene „mode“ gesetzt.

Prozedur copywindow(window)

Diese kleine Prozedur kopiert das Fenster „window“ an die Stelle, die durch das bewegliche Fadenkreuz bestimmt wird. Hier ist besonders die innere Prozedur pset interessant. In ihr befindet sich als Farbe die Function pd. Ist der Modus „S“ aktiviert, wird das Fenster

```
{---------------------------------------------------------------------------}
{ Print IBM EGA-Bildschirm im Grafikmodus. Die var "planes" bestimmt die Bild-}
{ schirmspeicherebene(n) und "vertikal" definiert die Hoehe des benutzten    }
{ Bildschirmspeichers in scan lines, z.B. 405, und horizontal die Breite des  }
{ Bildschirmspeichers.                                                        }
{ Als Drucker wurde ein ITOH 8510 benutzt.                                    }
{---------------------------------------------------------------------------}
{ Filename: Prtscr.EGA. }
procedure prtscr(planes:byte;vertikal,horizontal:integer);
var
   esc,cmd,n3,n2,n1,n0,data,cr,lf:char;
   m3,m2,m1,m0,m                  :integer;
   hor,ver,hbit,bit,scan,shrcnt,i:integer;
   egabytes                       :  array[0..7] of byte;
   prtbyte,prtbit,plane,mask,bt  :byte;
   vbyte,rbits,lp,col             :integer;
begin
{----------------------ITOH 8510 in Grafikmodus bringen--------------------}
esc:=chr(27);
cmd:=chr(99);        { software reset }
n0:=chr(49);
write(lst,esc,cmd,n0);   { zum Printer }
delay(2000);         { delay 2 sec }
```

Bild 6. Diese Prozeduren dienen zum Ausdruck der Grafiken

```
cmd:=chr(62);         { unidir }
write(lst,esc,cmd);   { zum Printer }

cmd:=chr(90);         { nur CR (kein LF) }
n0:=chr(128);
n1:=chr(0);
write(lst,esc,cmd,n0,n1); { zum Printer }

cmd:=chr(84);         { N/144 inch Line Feed Pitch }
n1:=chr(49);
n0:=chr(54);          { 8 * 2/144 inch }
write(lst,esc,cmd,n1,n0); { zum Printer }

if horizontal > 640   { Bild 90 Grad gedreht drucken }
then begin
     m:=vertikal;
     n1:=chr(78);      { Printer in Pica Modus       = max  640 pel/Druckzeile}
     if m > 640
     then n1:=chr(69); { Printer in Elite Modus      = max  768 pel/Druckzeile}
     if m > 768
     then n1:=chr(81); { Printer in compressed Modus = max 1088 pel/Druckzeile}
     write(lst,esc,n1); { zum Printer }
     end
else
     m:=horizontal;

{ Umwandlung von integer in ASCII }
m3:=48;               { ASCII 0 }
while m >= 1000
do begin
   m :=m-1000;
   m3:=m3+1;
   end;
m2:=48;
while m >= 100
do begin
   m :=m-100;
   m2:=m2+1;
   end;
m1:=48;
while m >= 10
do begin
   m :=m-10;
   m1:=m1+1;
   end;
m0:=48+m;

cmd:=chr(83);         { Bit image Graphics }
n3:=chr(m3);          { i.e.   0         }
n2:=chr(m2);          { i.e.   6         }
n1:=chr(m1);          { i.e.   4         }
n0:=chr(m0);          { i.e.   0         }

cr:=chr(13);          { CR }
lf:=chr(10);          { LF }
scan:=0;              { Bildschirmanfang  }

{-----------------------------------------------------------------------------}
{  Es wird immer mit 8 Nadeln gedruckt (bis auf den eventuellen Rest,am Ende).}
{  Ist x = 640, (der normale Bildschirm), so wird das Bild von oben nach unten}
{  gedruckt. Ist die Horizontale groesser als 640, so wird das Bild um 90 Grad}
{  im Uhrzeigersinn gedreht und dann gedruckt.                                }
{  Note: Fuer die Anpassung an andere Drucker ist darauf zu achten, dass beim }
{        hier benutzten ITHO Drucker im Grafikmodus die oberste Nadel die nie-}
{        derwertigste Stelle im Byte ist (im Gegensatz z.B. zu IBM und Epson).}
{        !!!! ITHO !!!! markiert die Stellen.                                 }
{-----------------------------------------------------------------------------}
planes := planes and $0F;            { nur 0..15 ist gueltig }

if horizontal = 640
then begin
     vbyte:=vertikal div 8;
     rbits:=vertikal-(vbyte*8);
     if rbits=0
     then begin
          vbyte:=vbyte-1;
          rbits:=8;
          end;
{-----Printer Scan Loop---- x = 640 horizontal---------------------------------}
     for i:= 0 to vbyte
     do begin
        if i=vbyte
        then lp:=rbits-1
        else lp:=7;
        mask := 1;
        plane := 0;

     while mask < 9                     { max 4 scans }
     do begin
        if planes and mask <> 0
        then begin
             write(lst,esc,cmd,n3,n2,n1,n0); { Printer Grafiks Operation }

             speicherebeneread(plane);     { Speicherebene selekt.  }

             for hor:=0 to 79              { 80 Bytes horizontal    }
             do begin
                for ver:=0 to lp           { 8 Bytes vertikal      }
                do                         { Speicherebene lesen   }
                egabytes[ver]:=mem[$a000:scan+hor+ver*80];

                for hbit:=0 to 7           { 8 bits horizontal/Byte }
                do begin
                   prtbyte:=0;
                   shrcnt:=7;              {!!!! ITHO !!!!}
                   for bit:=0 to lp        { 8 bits vertical/Printer-scan }
                   do begin
                      prtbyte:=prtbyte or ((egabytes[bit] and $80) shr shrcnt);
                      egabytes[bit]:=egabytes[bit] shl 1;
                      shrcnt:=shrcnt - 1;  {!!!! ITHO !!!!}
                      end;
                   write(lst,chr(prtbyte)); { Byte drucken }
                   end;
                end;
                write(lst,cr);              { CR nach 640 bytes }
             end;
             mask := mask shl 1;
             plane := plane + 1;            { nchste Speicherebene }
        end;
        scan:=scan + 640;                   { scan um 8 * 80 = 640 erhoehen }
        write(lst,lf);                      { LF zur nchsten Printerzeile }
     end;
  end
{-----Printer Scan Loop---- x > 640 horizontal--(sideway)---------------------}
else begin
     horizontal:=horizontal div 8;
     for scan := 0 to horizontal-1
     do begin
        mask := 1;
        plane := 0;

        while mask < 9                      { max 4 scans }
        do begin
            if planes and mask <> 0 then
            begin
               speicherebeneread(plane);
               write(lst,esc,cmd,n3,n2,n1,n0); { Printer Grafiks Operation }
               for col := vertikal-1 downto 0
               do begin
                  prtbyte:= mem[$A000:(col*horizontal)+scan];
                  bt:=0;                   {!!!! ITHO !!!!}
                  if (prtbyte and $80) = $80 then bt:=bt or $01;
                  if (prtbyte and $40) = $40 then bt:=bt or $02;
                  if (prtbyte and $20) = $20 then bt:=bt or $04;
                  if (prtbyte and $10) = $10 then bt:=bt or $08;
                  if (prtbyte and $08) = $08 then bt:=bt or $10;
                  if (prtbyte and $04) = $04 then bt:=bt or $20;
                  if (prtbyte and $02) = $02 then bt:=bt or $40;
                  if (prtbyte and $01) = $01 then bt:=bt or $80;
                  write(lst,chr(bt));
                  end;
               write(lst,cr);
            end;
            mask := mask shl 1;
            plane := plane + 1;  { nchste Speicherebene }
          end;
        write(lst,lf);             { LF zur nchsten Printerzeile }
        end;
end;

cmd:=chr(99);         { software reset des ITHO Printers }
n0:=chr(49);
write(lst,esc,cmd,n0);

speicherebeneread(0); { Read Mode 0 }
end;
```

kopiert. Ist der Modus „R“ aktiv, wird das Feld von der Ausdehnung „Fenster“ gelöscht. Hat man z. B. das Fenster an eine bestimmte Stelle kopiert, will es aber dort doch nicht haben, so kann man den Modus „R“ einschalten und noch einmal die gleiche Kopiertaste drücken.

Prozedur xyline

Mit dieser Prozedur wird der Startpunkt für eine Linie festgelegt (Shift-F6) und ein Fadenkreuz als Markierung in die Speicherebene „mode“ gesetzt.

Prozedur drawline

Mit dieser Prozedur kann eine Linie vom Startpunkt (xyline) bis zum Standort des beweglichen Fadenkreuzes gezogen werden. Die Grafik in Bild 2 wurde mit diesen beiden Prozeduren gemacht. Der darüberliegende Text ist danach mit den Textfunktionen „G“, „i“ und „o“ geschrieben worden.

Prozedur clearplane

Mit der Prozedur clearplane wird oder werden die selektierte(n) Speicherebene(n) gelöscht.

Palette Register Initialisierung

Es werden alle 16 Palette-Register benutzt. Sie sind wie folgend aufgeführt initialisiert:

Bildschirm Monochrom/Farbe
1. Speicherebene C0, normal video/rot
2. Speicherebene C1, normal video/blau
3. Speicherebene C0 und C1, intensified video/cyan
4. Fadenkreuz C2, normal video/rosa
5. Fadenkreuz und C0 oder C1, intensified video/gelb
6. Fadenkreuz und C0 und C1, normal video/gelb
7. Modus C3, normal video/gelb

Damit ist sichergestellt, daß das Fadenkreuz immer sichtbar ist. Durch die Tatsache, daß bei einer Überlappung von Zeichnung und Fadenkreuz eine andere Farbe entsteht, läßt sich selbst bei kleinen Objekten eine genaue Positionierung des Fadenkreuzes erreichen. Besonders hier zeigt es sich, daß für die Bewegung des Fadenkreuzes die Cursortasten in Verbindung mit der Bewegungszahl der Maus überlegen sind.

prtscr(plane,vertikal,horizontal)

Die Prozedur prtscr (*Bild 6*) ist in der Lage, alle vorkommenden Bildformate auf einem ITOH 8510 SP zu drucken. Für andere Drucker muß die Prozedur angepaßt werden. Ist die Anzahl der horizontalen Punkte größer als 640, so wird das Bild um 90 Grad im Uhrzeigersinn gedreht und dann gedruckt. Ist die Anzahl der vertikalen Punkte nach der Drehung immer noch größer als 640, wird der Drucker entweder auf 768 Punkte bzw. 1088 Punkte pro Zeile umgeschaltet.

Leiterplattenverdrahtung mit EGA-TCAD.

Bei der Verdrahtung von Leiterplatten ist es wichtig, daß man vorher berücksichtigt, welche Nadelabstände der Drucker hat. Beim ITOH 8510 SP beträgt der horizontale Abstand 1/80 Zoll und der vertikale Abstand 1/72 Zoll. Der Pin-Abstand von IC-Sockeln beträgt 1/10 Zoll, so daß auf dem Bildschirm ein Pin-Abstand von 8 Punkten (8/80 = 1/10 Inch) gewählt werden muß, um eine 1:1 Darstellung zu erreichen. Bei einem Quadrat von 5 × 5 Punkten für die Lötösen kann zwischen den Lötösen gerade noch eine Leitung durchgeführt werden, was normalerweise ausreicht. Da der Abstand zwischen den Pin-Reihen eines IC nicht so kritisch ist, sollte man dafür sorgen, daß dieser Abstand immer in vertikaler Richtung des Druckerbildes verläuft.

Ob nun Glückwunschkarten, Leiterplatten oder sonstige Grafik, das Arbeiten mit der EGA und diesem Programm macht Spaß. (Bild 2 entstand in weniger als 20 Minuten).

Abschließend sei gesagt, daß die IBM-EGA mit Sicherheit ein Vorreiter in der gehobenen Grafikszenerie der PCs war und ist. Das zeigt die Tatsache, daß sie unter den vielen „Clones“ schon fast eine Seltenheit geworden ist.

Was nun die VGA in der neuen Serie der IBM PS/2 bietet, zeigt schon mal eines: sie ist EGA-kompatibel und hat vor allem Bildschirme mit square pixeln, also keine Verzerrungen mehr.

```
{---------------------------------------------------------------------------}
{                  Funktions Tasten fuer das Programm EGA_TCAD               }
{ Ulrich Cebulla                                                             }
{ 7036 Schoenaich, Hermann Werner Str. 6                     20.05.1987      }
{---------------------------------------------------------------------------}

procedure helptext(horizontal,vertikal:integer);

begin
wrtinvert:= true;
fontpointer(false);
gotoxy(20,1);
writeln('   x=',horizontal+1,         E G A - T C A D        y=',vertikal+1,'   ');
wrtinvert:= false;
fontpointer(true);
gotoxy(1,3);
writeln('---Taste----------Funktion-----------------------------------------------');
writeln('   H/h =>         Tastenfunktionen / Modus anzeigen, ein/aus');
writeln('   E  =>          Programmende.');
writeln('   L  =>          Bildschirmspeicher von der Diskette laden.');
writeln('   S  =>          Bildschirmspeicher auf die Diskette speichern.');
writeln('   M  =>          Bild auf dem Bildschirm rotieren, ein/aus. (Cursor Tasten)');
writeln('   C  =>          Bildschirm scrollen, oben/unten/links/rechts. (Cursor Tasten)');
writeln('   P  =>          Bildschirm auf dem Grafikdrucker drucken. (ITHO 8510 SP)');
writeln('   d  =>          zeichnen, ein/aus.');
writeln('   r  =>          Punkt setzen = !S!, Punkt loeschen = !R!, (ein/aus).');
writeln(' cursor  =>       Fadenkreuz nach links/rechts/oben/unten bewegen.');
writeln('   F,x   =>       Fadenkreuzes finden; x/y Position anzeigen, ein/aus.');
writeln('1..9,0,*,+,- =>   Bewegungszahl des Fadenkreuzes, Punkte/cursor Taste.');
writeln('   F1 =>          Zeichenebene C0.');
writeln('   F2 =>          Zeichenebene C1.');
writeln('   F3 =>          Zeichenebene C0 und C1.');
writeln('   F6 =>          Linie von x/y (shift F6) nach x/y von Fadenkreuz ziehen.');
writeln('   Z  =>          Zeichenebene Cx loeschen.');
writeln(' F7..F10 =>       Selektion/Kopie Fenster 0..3 (shift/ctrl).');
writeln('   T  =>          Textmodus einschalten. Textende = Eingabetaste.');
writeln('   G  =>          Scalefaktor fuer Text um 1 erhoehen.');
writeln('   g  =>          Scalefaktor fuer Text um 1 erniedrigen.');
writeln('   o  =>          Charakterbox im Textmodus loeschen.');
writeln('   f  =>          Fontumschaltung von 8x8 nach 8x14 und zurueck.');
writeln('   v  =>          Text vertikal schreiben, ein/aus.');
writeln('   i  =>          Text invertiert schreiben, ein/aus.');
writeln('-------------------------------------------------------------------------');
writeln;
writeln('              zurueck =>    Taste "H"');
end;
```

Bild 7. Diese Prozedur zeigt die Belegung der Tasten an

Josef Schmucker

Echtzeituhr im ROM-Sockel

Ein Uhrenmodul im Atari ST sorgt für genaue Zeiten

Das Prinzip der hier beschriebenen Hardware-Uhr beruht auf der im mc 12/86 beschriebenen Smart-Watch DS1216. Der hauptsächliche Unterschied zu dieser besteht darin, daß das Uhrenmodul DS1216E für den Betrieb mit ROM-Speichern geeignet ist.

Der kritische Betrachter wird sofort sagen, daß bei ROM's nur das Lesen der Uhrzeit möglich sein wird, da bei ROM's kein Schreibzugriff möglich ist. Das Gegenteil beweist das hier beschriebene Verfahren.

Zunächst soll jedoch noch einmal auf das grundsätzliche Arbeitsprinzip der Uhrenmodulserie DS1216 eingegangen werden. Die Uhr verhält sich gegenüber dem Computersystem, in dem das Modul eingesetzt wird, neutral. Die Steuerleitungen (Adreß-, Daten- und Select-Leitungen) werden bis auf das Chip-Enable-Signal direkt zum entsprechenden Speicherbaustein weitergeleitet (*Bild 1*). Wenn jedoch auf bestimmte Speicherzellen in einer gewissen Reihenfolge zugegriffen wird, greift das Uhrenmodul ein und unterbricht die Chip-Enable-Leitung. Nun kann man durch eine bestimmte Anzahl von Speicherlesezyklen (in diesem Fall 64) die Uhr setzen oder lesen. Durch die hier angewandte Methode beträgt die Wahrscheinlichkeit, daß ein normaler Speicherzugriff die Uhr aktiviert, 1 zu 10^{19}.

Das beschriebene Prinzip wird nun am Beispiel des DS1216E in die Tat umgesetzt. Als Ausgangszustand wird der normale Betrieb angenommen. Das heißt, die Uhr verhält sich dem System gegenüber neutral. Erfolgen nun direkt aufeinanderfolgend 64 Speicherlesezyklen auf den Baustein, unter dem das Uhrenmodul sitzt, bei denen die Adreßleitung A2 LOW-Zustand hat und die Adreßleitung A0 der Reihe nach die Zustände einer vorgegebenen Bitfolge annimmt, so nimmt das Uhrenmodul an, daß es jetzt an der Reihe ist und unterbricht die Chip-Enable-Leitung. Die Bitfolge ist in *Bild 2* dargestellt. Sie stellt gewissermaßen den Schlüssel zur Uhr dar. Die nachfolgenden 64 Speicherlesezyklen lesen oder setzen die Uhr. Das Lesen geschieht auf folgende Weise: Die Adreßleitung A2 wird auf HIGH-Pegel gehalten, das heißt für die Uhr, daß die nachfolgenden 64 Lesezyklen die Uhr lesen sollen. Auf D0 wird nun bei jedem Lesezugriff seriell ein Bit der Uhrzeit ausgelesen; beginnend vom Register 0, Bit 0 bis zum Register 7, Bit 7. Danach wird die Uhr von selbst dem System gegenüber inaktiv und tut so, als wäre nichts gewesen. Das Schreiben geschieht auf ähnliche Weise. Nur mit dem Unterschied, daß die einzelnen Bits der Information der Reihe nach auf eine Adreßleitung beim Lesen gelegt werden. Zum Schreiben wird demnach das Adreßbit A2 auf LOW-Pegel gehalten und die Uhreninformation wiederum seriell von Register 0, Bit 0 beginnend bis zu Register 7, Bit 7 auf das Adreßbit A0 beim Lesen gelegt. Danach wird die Uhr wie beim Lesen inaktiv.

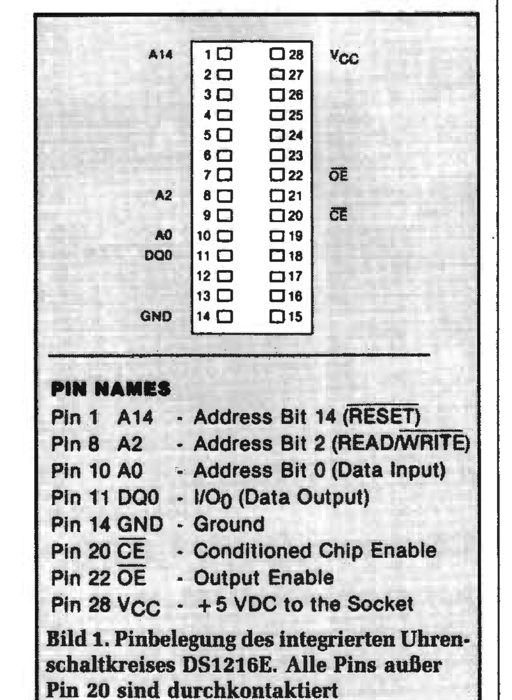


Bild 1. Pinbelegung des integrierten Uhrenschaltkreises DS1216E. Alle Pins außer Pin 20 sind durchkontaktiert

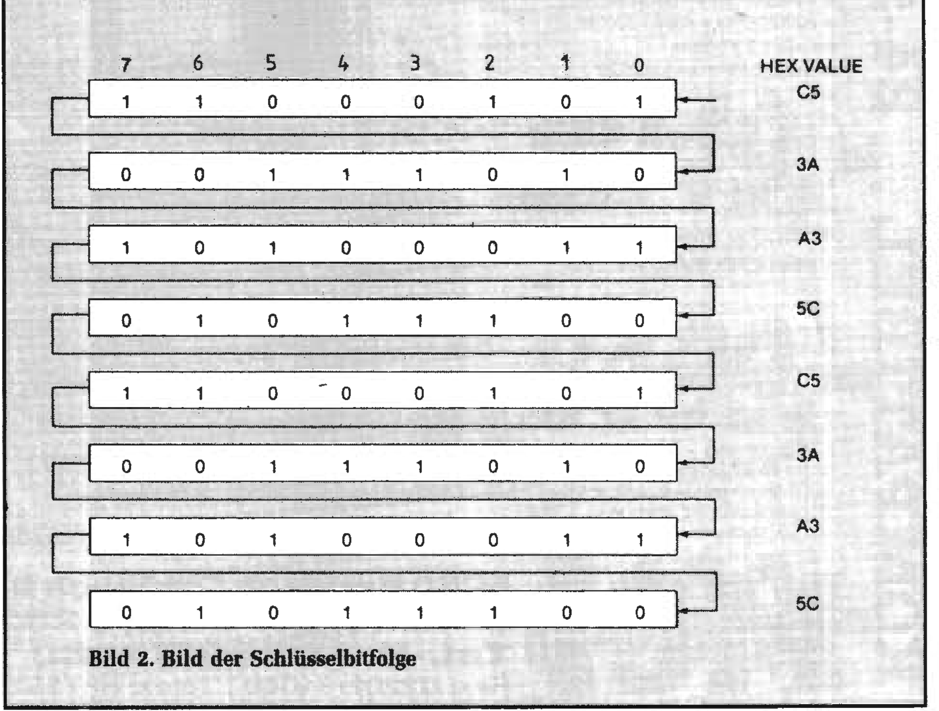


Bild 2. Bild der Schlüsselbitfolge

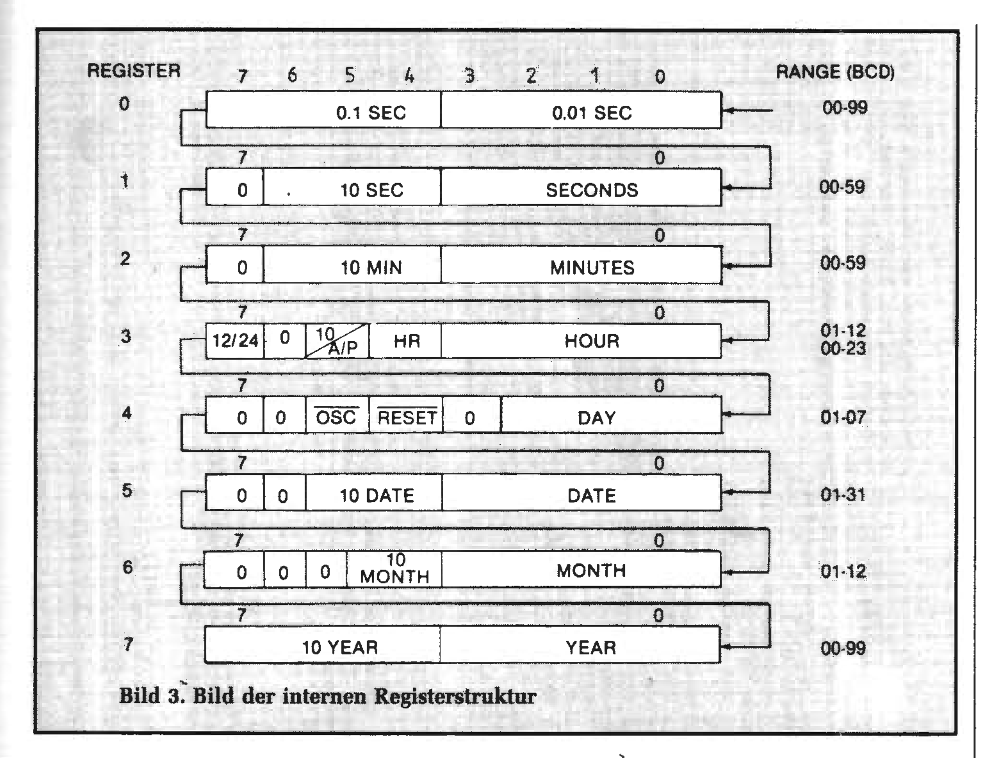


Bild 3. Bild der internen Registerstruktur

Zusammenfassend kann man sagen, daß bei einem Zugriff auf die Uhr immer zuerst die Schlüsselfolge gelesen werden muß und daraufhin die Lese- bzw. „Schreib"-Operation erfolgt. Die Information der Uhr ist intern in acht 8-Bit-Registern im BCD-Format abgelegt. Die Struktur der acht Register ist aus *Bild 3* zu ersehen. Mit Bit 7 im Register 3 kann zwischen 12 und 24-Stunden-Mode umgeschaltet werden. Im Register 4 sollte in den oberen 4 Bit immer 0001 stehen.

Einbau in den Atari 520 ST

Im Vergleich zu anderen auf dem Markt in vielfältiger Weise angebotenen Möglichkeiten, eine Uhr in den Atari ST zu implementieren, stellt diese Lösung die mit dem besten Preis-Leistungs-Verhältnis dar. Während andere Lösungen eine Platine (mit Uhren-IC, PAL, Akku usw.) benötigen oder gar den Keyboard-Prozessor mit großen Akkus puffern (ganz abgesehen davon, daß die Tastaturuhr relativ ungenau geht), paßt sich dieses Modul, indem es nur die gleichen Grundmaße wie ein ROM-IC einnimmt, elegant in das System ein. Beim Atari ST gibt es grundsätzlich zwei Möglichkeiten, diesen Uhren-Chip einzusetzen: entweder in einem der System-ROM-Steckplätze oder auf einer EPROM-Steckplatine für den Modulschacht. Entsprechende Platinen sind im Handel erhältlich. Im allgemeinen ist aber der interne Einbau der externen Lösung vorzuziehen. In den Atari 1040 ST kann man das etwa 9 mm hohe Modul nicht einbauen, da in diesem Modell über den ROM-ICs nur etwa 2 mm Raum frei ist. Beim Atari 1040 ST muß also die externe Lösung angewendet werden.
Betrachtet man nun den Atari 520 ST von innen, so kommt eigentlich aus Platzgründen (Bauhöhe) nur der Steckplatz U2 in Frage. Auf diesem Sockel wird das Modul immer unter geraden Adressen angesprochen.

Betriebssoftware für den Atari ST

Die hier vorgestellten Programme sind vollständig in C geschrieben und müßten auf jedem C-Compiler, der nach Kernigham/Ritchie arbeitet, übersetzbar sein. Das Programm „uhr.c" (*Bild 4*) dient (über ein kleines Menü) zum Stoppen und Setzen der Hardware-Uhr und zum Auslesen der Zeitinformation aus dem Uhrenmodul mit einem impliziten Setzen der Atari-Systemuhr. Das Programm „setuhr.c" (*Bild 5*) stellt eine Untermenge von „uhr.c" dar, und ist für den Auto-Ordner gedacht, um beim Einschalten des Systems die Atari-Uhr zu setzen.

Das Stoppen der Hardware-Uhr wird durch Setzen des Bits 5 in Register 4 erreicht. Dazu wird zuerst der Uhreninhalt gelesen und im Uhrenpuffer (watch_buf) abgelegt. Danach wird das Stopp-Bit gesetzt und der Uhrenpuffer in die Uhr zurückgeschrieben. Um die Uhr zu setzen, wird die Zeit und das Datum vom Programm erfragt. Nach den Eingaben werden die Daten in das Uhrenformat umgewandelt, im Uhrenpuffer abgelegt und in die Uhr geschrieben. Dabei werden die Zehntelsekunden und der Wochentag immer auf einen festen Wert gesetzt, da diese Werte für diesen Fall irrelevant sind. Beim Lesen der Uhreninformationen werden die uhrinternen Register nach dem Lesen im Uhrenpuffer abgelegt. Danach werden sie in das

```
/*************************************************************************/
/*                                                                       */
/*      Programm zum Bearveiten der Hardwareuhr DS1216E von DALLAS-      */
/*      SEMICONDUCTOR im Atari-ST.                                       */
/*      UHR.C                                                            */
/*                                                                       */
/* 01.03.87 V 1.0: Josef Schmucker, Theresienhoehe 6, 8000 Muenchen 2   */
/*************************************************************************/
#include <osbind.h>
#include <stdio.h>

#define RBASE 0xFE0008L                    /* Adressen fuer Sockel U2 im   */
#define WBASE 0xFE0000L                    /* Atari.                       */
#define KEY   0x5CA33AC5L;                 /* KEY muss zur Uhr geschrieben */
                                           /* werden, damit ROM disabled   */
#define begin {                            /* wird.                        */
#define end }                              /* Redefines fuer besseres C    */
#define then

char watch_buf[8], timebuf[30];
int  betriebsart,time,date;
char texte[6][38] = begin
                    " Tag     (00..31)_______________: ",
                    " Monat   (01..12)_______________: ",
                    " Jahr    (80..99)_______________: ",
                    " Stunde  (00..23)_______________: ",
                    " Minute  (00..59)_______________: ",
                    " Sekunde (00..59)_______________: ",
                  end;
enable_watch ()                           /* Uhr durch Lesen der Vergleichs */
begin                                     /* adressen aktivieren. Achtung   */
  int i,j;                                /*dadurch wird das ROM deaktiviert*/
  char *adr,cdummy;                       /* Deshalb sollte die Zeit vom    */
```

Bild 4. Das Programm Uhr.c dient zum Einstellen der Uhrzeit ▶

```
  register long y;                     /* aktivieren der Uhr bis zum      */
  register long x;                     /*Lesen bzw. Schreiben so kurz wie*/
                                       /* nur moeglich gehalten werden.   */
  cdummy = *(char*)(RBASE);            /* Dummy-Read zum Ruecksetzen      */
  for (i=0; i<2; i++)
  begin
    x = KEY;
    for (j=0; j<32; j++)
    begin
      y = (x & (long)0x01);
      y <<= 1;                         /* statt 'y = y <<1' wegen Geschw.*/
      adr = (char*)(WBASE + y);
      cdummy = *adr;
      x >>= 1;
    end
  end
end


char ascii_to_bcd (l,r)                /* Umwandlung von 2 Ascii-Chars in*/
char l, r;                             /* ein BCD-Char.                   */
                                       /* Beim Aufruf muss in l der linke*/
begin                                  /* und in r der rechte Teil der    */
  char o;                              /* BCD-Zahl stehen.                */
                                       /* Eine Gueltigkeitsueberpruefung */
  o = l & 0x0F;                        /* erfolgt nicht.                  */
  o = (o << 4);                        /*                                 */
  o = o | (r & 0x0F);                  /*                                 */
  return (o);                          /*                                 */
end
int bcd_to_int (c)                     /* Umwandlung einer BCD-Zahl in    */
char c;                                /* eine INTEGER-Zahl.              */
begin                                  /* Eine Ueberpruefung auf Gueltig-*/
  int x;                               /* keit erfolgt nicht.             */
  char c1;

  c1 = c;
  x = (int)(c1 & 0x0F);
  c1 >>= 4;
  c1 = c1 & 0x0F;
  x = x + ((int)c1 * 10);
  return (x);
end


read_watch ()                          /* Lesen der Hardware-Uhrzeit, die */
begin                                  /* in watch_buf abgespeichert wird */
  int i,j;
  char x;

  Jdisint (5);                         /* 200Hz-Interrupt sperren         */
  enable_watch ();                     /* ROM sperren und Uhr freigeben   */
  for (i=0; i<8; i++)
  begin
    for (j=0; j<8; j++)
    begin
      x = 0;
      x = *(char*)RBASE;
      x = x & 0x01;
      x <<= j;
      watch_buf[i] = watch_buf[i] | x;
    end
  end
  Jenabint (5);                        /* 200Hz-Interrupt freigeben       */
end
```

```
write_watch ()                         /* Schreiben der Uhrzeit, die in */
begin                                  /* watch_buf steht, in die Hard- */
  int i,j;                             /* wareuhr.                      */
  char x,cdummy;
  char *adr;

  Jdisint (5);                         /* 200Hz-Interrupt sperren       */
  enable_watch ();
  for (i=0; i<8; i++)
  begin
    for (j=0; j<8; j++)
    begin
      x = watch_buf[i];
      x >>= j;
      x &= 0x01;
      x <<= 1;
      adr = (char*)(WBASE + (long) x);  /* Schreiben durch Lesen        */
      cdummy = *adr;
    end
  end
  Jenabint (5);                         /* 200Hz-Interrupt freigeben    */
end


dallas_to_atari ()                     /* Konvertieren der Hardware-Uhr- */
begin                                  /* daten in das ATARI-Format.     */
  int seconds,minutes,hours;           /* Durch Schieben wird der je-    */
  int day,month,year;                  /* weilige Integer-Wert in die    */
                                       /* richtige Position gebracht.    */
  seconds = bcd_to_int (watch_buf[1]);
  seconds = seconds / 2;
  minutes = bcd_to_int (watch_buf[2]);
  minutes = (minutes << 5);
  hours   = bcd_to_int (watch_buf[3]);
  hours   = (hours << 11);
  day     = bcd_to_int (watch_buf[5]);
  month   = bcd_to_int (watch_buf[6]);
  month   = (month << 5);
  year    = bcd_to_int (watch_buf[7]);
  year    = (year - 80);
  year    = (year  << 9);
  date = day + month + year;
  time = seconds + minutes + hours;
end


input_time ()                           /* Eingeben der aktuellen Uhr- */
begin                                   /* zeit in watch_buf           */
  int  i;
  char j;

  printf (" Bitte geben Sie die aktuelle Zeit ein --------------->\n");
  printf (" Achtung! Die Eingabe muss immer 2-stellig erfolgen");
  printf (" z.b. 01 oder 12)\n\n");
  for (i=0; i<6; i++)
  begin
    do
    begin
      printf ("%s",&texte[i][0]);
      scanf  ("%s",&timebuf[i*2]);
      j = atoi (&timebuf[i*2]);
    end
    while ((j < 0) || (j > 99));
  end
end
```

```
set_dallas ()                               /* Hardwareuhr stellen       */
begin
  input_time ();                            /* Zeit einlesen und in      */
  watch_buf[0] = 0x00;                      /* Atari-Format umwandeln    */
  watch_buf[1] = ascii_to_bcd (timebuf[10],timebuf[11]);
  watch_buf[2] = ascii_to_bcd (timebuf[8],timebuf[9]);
  watch_buf[3] = ascii_to_bcd (timebuf[6],timebuf[7]);
  watch_buf[4] = 0x01;
  watch_buf[5] = ascii_to_bcd (timebuf[0],timebuf[1]);
  watch_buf[6] = ascii_to_bcd (timebuf[2],timebuf[3]);
  watch_buf[7] = ascii_to_bcd (timebuf[4],timebuf[5]);
  write_watch ();
end


deactivate_clock ()                  /* Stoppen der Hardwareuhr, durch     */
begin                                /* setzen eines Hardwareuhr - Flags   */
  read_watch ();
  watch_buf[4] = watch_buf[4] | 0x20;
  write_watch ();
  printf ("\n Die Uhr wurde gestoppt !!!\n\n");
end


set_atari ()                         /* Setzen der ATARI-Uhr durch die     */
begin                                /* Informationen der Hardwareuhr      */
  read_watch();
  dallas_to_atari ();
  Tsettime (time);
  Tsetdate (date);
end

main()                               /* Beginn des Hauptprogramms          */
begin
  Super (0L);                        /* Berechtigung zum Lesen der Adresse */
  betriebsart = 1;                   /* 0xFE0000 erwerben, sonst Bomben!!! */
  while (betriebsart != 0)           /* Schleife bis Programmende          */
  begin
    printf ("\n Bearbeitungsprogramm fuer die Hardwareuhr DS1216E\n");
    printf (" auf der Adresse %lx      V1.0    Josef Schmucker\n\n",WBASE);
    printf (" Programmende ..................................( 0 )\n");
    printf (" Hardwareuhr deaktivieren ......................( 1 )\n");
    printf (" Hardwareuhr setzen ............................( 2 )\n");
    printf (" ATARI-Uhr ueber Hardwareuhr setzen ............( 3 )\n\n");
    printf (" A u s w a h l ---------------------------------------> ? ");
    scanf ("%d",&betriebsart);
    switch (betriebsart)
    begin
      case 0:
             break;
      case 1:
             deactivate_clock ();
             break;
      case 2:
             set_dallas ();
             break;
      case 3:
             set_atari ();
             break;
      default:
             printf ("\n FALSCHE Auswahl !!!!!!!!!!!!!!!!\n\n\n");
             break;
    end
  end
end
```

```
/***********************************************************************/
/*                                                                     */
/*      Programm zum Setzen der ATARI-Uhr ueber die Hardwareuhr        */
/*      DS 1216 E  von DALLAS-SEMICONDUCTOR                            */
/*      SETUHR.C                                                       */
/*                                                                     */
/*  01.03.87 V 1.0: Josef Schmucker, Theresienhoehe 6, 8000 Muenchen 2 */
/***********************************************************************/
#include <osbind.h>
#include <stdio.h>

#define RBASE 0xFE0008L              /* Adressen fuer Sockel U2 im   */
#define WBASE 0xFE0000L              /* Atari.                       */
#define KEY   0x5CA33AC5L            /* KEY muss zur Uhr geschrieben */
                                     /* werden, damit ROM disabled   */
#define begin {                      /* wird.                        */
#define end }                        /* Redefines fuer besseres C    */
#define then

char watch_buf[8];
int  time,date;


enable_watch ()                      /*Uhr durch Lesen der Vergleichs- */
begin                                /*adressen aktivieren. Achtung    */
  int i,j;                           /*dadurch wird das ROM deaktiviert*/
  char *adr,cdummy;                  /*Deshalb sollte die Zeit vom     */
  register long y;                   /*Aktivieren der Uhr bis zum      */
  register long x;                   /*Lesen bzw. Schreiben so kurz wie*/
                                     /*nur moeglich gehalten werden.   */
  cdummy = *(char*)(RBASE);
  for (i=0; i<2; i++)
  begin
    x = KEY;
    for (j=0; j<32; j++)
    begin
      y = (x & (long)0x01);
      y <<= 1;
      adr = (char*)(WBASE + y);
      cdummy = *adr;
      x >>= 1;
    end
  end
end


int bcd_to_int (c)                   /*Umwandlung einer BCD-Zahl in eine */
char c;                              /*INTEGER-Zahl.                     */
begin                                /*Eine Ueberpruefung auf Gueltigkeit*/
  int x;                             /*erfolgt nicht.                    */
  char c1;

  c1 = c;
  x = (int)(c1 & 0x0F);
  c1 = (c1 >> 4);
  c1 = c1 & 0x0F;
  x = x + ((int)c1 * 10);
  return (x);
end


read_watch ()                        /*Lesen der Hardware-Uhrzeit, die */
begin                                /*in watch_buf abgespeichert wird */
  int i,j;
  char x;
```

Bild 5. Mit diesem Programm wird die Atari ST-Uhr gesetzt

```
  Jdisint (5);                              /*200Hz-Interrupt sperren          */
  enable_watch ();                          /*ROM sperren und Uhr freigeben    */
  for (i=0; i<8; i++)

  begin
    for (j=0; j<8; j++)
    begin
      x = 0;
      x = *(char*)RBASE;
      x = x & 0x01;
      x <<= j;
      watch_buf[i] = watch_buf[i] | x;
    end
  end
  Jenabint (5);                             /*200Hz-Interrupt freigeben        */
end


dallas_to_atari ()                          /* Konvertieren der Hardware-Uhr*/
begin                                       /* daten in das ATARI-Format     */
  int seconds,minutes,hours;                /* Durch Schieben wird der je-   */
  int day,month,year;                       /* weilige Integer-Wert in die   */
                                            /* richtige Position gebracht.   */
  seconds = bcd_to_int (watch_buf[1]);
  seconds = seconds / 2;
  minutes = bcd_to_int (watch_buf[2]);
  minutes = (minutes << 5);
  hours   = bcd_to_int (watch_buf[3]);
  hours   = (hours << 11);
  day     = bcd_to_int (watch_buf[5]);
  month   = bcd_to_int (watch_buf[6]);
  month   = (month << 5);
  year    = bcd_to_int (watch_buf[7]);
  year    = (year - 80);
  year    = (year  << 9);
  date = day + month + year;
  time = seconds + minutes + hours;
end


main ()                              /* Beginn des Hauptprogramms            */
begin                                /*                                      */
  Super (0L);                        /* Berechtigung zum Lesen der Adresse */
  read_watch();                      /* 0xFE0000 erwerben, sonst Bomben !! */
  dallas_to_atari ();
  Tsettime (time);                   /* Atari-Uhr setzen                     */
  Tsetdate (date);
end
```

Atari-Format umgewandelt und die Atari-Zeit durch die Aufrufe „Tsettime" und „Tsetdate" gesetzt. Die Uhr ist den Programmen unter der Adresse 0XFE0000 bekannt, wenn diese im Sokkel U2 eingesteckt wird. Bei einem Betrieb am ROM-Modulport müssen im Programm die Adreßdefinitionen von FExxxx auf FAxxxx bzw. FBxxxx geändert werden.

Eine Besonderheit, die im Atari ST auftritt ist, daß der Sockel, in dem das Modul steckt, nur bei jeder zweiten Speicheradresse angesprochen wird, da der Speicher in UPPER- und LOWER-Speicherbänken organisiert ist. Dies muß bei der Erstellung der entsprechenden Routinen berücksichtigt werden. Ansonsten ist der Source-Code der einzelnen Routinen selbsterklärend.

Nur wenig Aufwand

Der Einbau in den Atari 520 ST ist unproblematisch. Nachdem das äußere Kunststoffgehäuse abgeschraubt worden ist, werden mit einer Flachzange die Blechlaschen am Rand des inneren Blechgehäuses gelöst. Danach kann die obere Deckelhälfte nach oben weggeklappt werden. Nun wird das ROM mit der Steckplatzbezeichnung „U2" mit einem entsprechenden Werkzeug herausgehebelt und das Uhrenmodul an dessen Platz eingesetzt. Danach ist nur noch das ROM auf das Modul aufzustecken, und sie besitzen einen Atari ST mit einer netzausfallsicheren Hardware-Uhr. Beim Einbau sollten sie ein besonderes Augenmerk auf die Gehäusemarkierungen der Bausteine und auf einen einwandfreien Steckkontakt der einzelnen Stifte legen. Geschlossen wird der Atari einfach in umgekehrter Reihenfolge. Nur die Blechlasche in der Höhe des U2-Sockels wird nicht geschlossen, da sonst das ROM mit dem Modul am Blechgehäuse anstoßen würde.

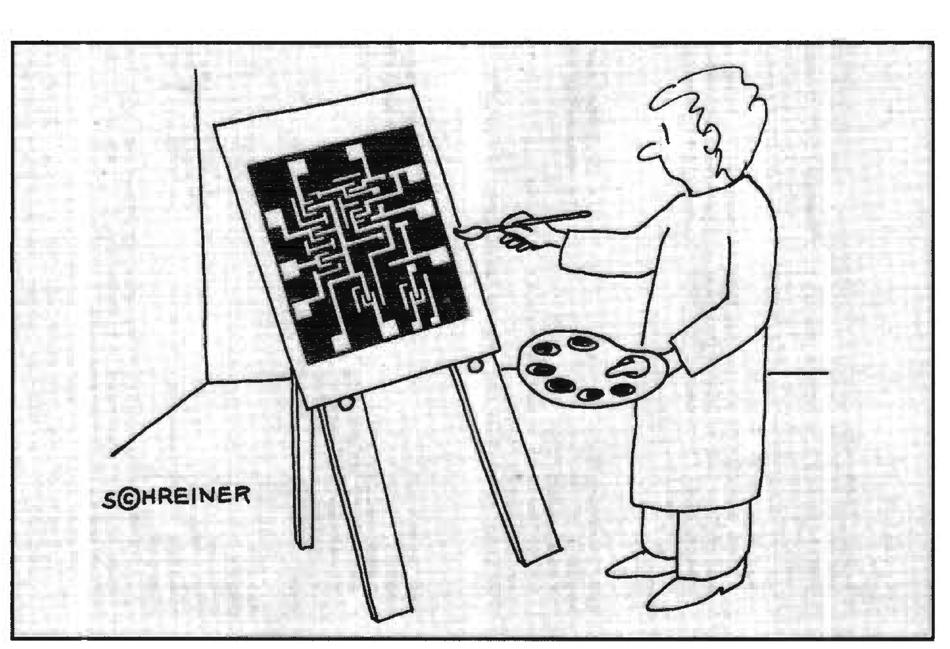
Implementierung auf anderen Computern

Die Implementierung in andere Computer als dem Atari ST ist grundsätzlich möglich. Jedoch sollten dabei einige Punkte berücksichtigt werden. Das Chip-Enable-Signal wird im normalen Betrieb immer um ca. 10 ns verzögert. In Systemen, in denen der Speicherzugriff sehr zeitkritisch gehalten ist, können hier eventuell Probleme entstehen. Des weiteren muß während eines Speicherzugriffs immer Output-Enable (OE) und Chip-Enable (CE) auf LOW-Pegel liegen, damit das Uhrenmodul einen vollständigen Speicherzugriff erkennt.

Literatur

[1] Datenblatt DS1216E von Dallas Semiconductor; Vertrieb Atlantik Electronic, München.

[2] *Eicher Michael*, Uhrzeit gut versteckt, mc 12/86, Seite 70.



Robert Kandlbinder

Backup ganz bequem

Umfangreiche „Directories“ sichern

Will man alle Dateien eines Platten-Directories auf Diskette sichern, dann wird man von MS/PC-DOS oft im Stich gelassen. Copy *.* kopiert nämlich nur so lange, bis die Zieldiskette voll ist. Eine komplette Kopie liefert das Programm COPYKOMP.

Grundsätzlich gibt es unter MS/PC-DOS zwei verschiedene Arten von Diskettenoperationen. Das sind zum einen die CP/M-kompatiblen und zum anderen die Unix-ähnlichen. Mit CP/M-Funktionen kann man keine Pfade verwenden. Das klassische Beispiel: WordStar 3.4. Unix-Funktionen dagegen unterstützen Pfade bis zu 128 Zeichen, stehen aber erst ab Version 2.10 zur Verfügung [1]. Das Programm COPYKOMP (*Bild*) verwendet deshalb die Unix-Funktionen. Es kopiert, wie der DOS-Befehl COPY, keine Hidden- und System-Dateien, diese können auch nicht mit dem Befehl DIR angezeigt bzw. mit ERASE *.* oder DEL *.* gelöscht werden. Wer trotzdem Hidden- und/oder System-Dateien kopieren will, muß für COPYKOMP in der Routine *find_first* die entsprechenden Attribute einsetzen. Alle kopierten Dateien erhalten aber das Archive-Attribut analog zum DOS-Befehl COPY.

Da die Funktionen 2F bis 57 (hex) erst ab DOS 2.10 zur Verfügung stehen, wird als erstes geprüft, ob die geladene DOS-Version gleich oder größer Version 2.10 ist. Wenn nicht, wird ein entsprechender Hinweis ausgegeben und COPYKOMP beendet. Ansonsten verzweigt das Programm zum Label *ok_vers*. Hier wird das Logo ausgegeben.

Wird ein Programm von der DOS-Ebene aus mit Parametern aufgerufen, werden diese im PSP ab Offset 81_h übergeben (max. 128 Zeichen), die Parameterlänge steht im Offset 80_h. Wird kein Parameter übergeben, steht im Offset 81_h das Carriage Return (CR). Dies wird ab Label *param_check* geprüft. Ohne Parameter wird nach *no_param* verzweigt, und nach Ausgabe der Fehlermeldung *err_msg_5* wird das Programm beendet.

Da das Programm standardmäßig als Ziellaufwerk A verwendet, muß geprüft werden, ob als Parameter das Laufwerk A übergeben wurde oder, wenn das Default-Laufwerk verwendet wird, ob es sich nicht um das Laufwerk A handelt. Denn für Quell- und Zieldatei wird ja der gleiche Name verwendet, und das ist auf einem Laufwerk nicht zulässig. Die entsprechende Prüfung erfolgt beim Label *lwa_test*. Da Parameter übergeben wurden, ist das erste Byte ein Leerzeichen (Blank). Deshalb werden das zweite und dritte Byte in AX geladen und auf „A:“ bzw. „a:“ geprüft. Verläuft der Test positiv, verzweigt das Programm zum Label *lwa_test2*. Trifft dies nicht zu, wird geprüft, ob überhaupt ein Laufwerk angegeben wurde. Wenn ja, steht in AH = „:“. Ist kein Doppelpunkt vorhanden, dann soll das Default-Laufwerk verwendet werden. Mit der Funktion 19_h wird deshalb das Default-Laufwerk abgefragt, ob es das Laufwerk A ist (0 = A). Ist eine der Bedingungen erfüllt, wird die Meldung err_msg_2 ausgegeben und nach der Bestätigung beim Label *lw_chg2* das Laufwerk für den Dialog *(default_lw)* und für den Namen der Zieldatei *(target_lw)* auf B geändert. Für die Zieldatei wird das Root-Verzeichnis (A:\ bzw. B:\) aus folgendem Grund verwendet: Ist vor dem Aufruf von COPYKOMP in A: oder B: ein Unterverzeichnis aktiv, dann würde sich nach dem Diskettenwechsel DOS melden, da das „gültige“ Unterverzeichnis auf der neuen Diskette fehlt! Nun müßte man erst das Verzeichnis wechseln und COPYKOMP erneut aufrufen.

Für Unix-Aufrufe muß DS:DX auf eine ASCII-Zeichenfolge zeigen, die den Laufwerk-, Pfad- und Dateinamen beinhaltet. Als Endemarkierung dient hierbei ein Null-Byte. Deshalb wird beim Label *lwa_nein* die Parameterlänge in BL geladen, der Offset im Code-Segment addiert und das letzte Zeichen (CR) durch 0 ersetzt. Denn dieser Parametertext ist später zur Suche nach dem ersten übereinstimmenden Eintrag erforderlich.

Nun gehen wir rückwärts *(test)* durch den Parametertext und suchen den Dateinamenanfang. Hierfür gibt es drei Möglichkeiten:

```
;Assembler = IBM-MASM   Version 2.0
LF          equu     10
CR          equ      13
param_lg    equ      80h          ;Längenbyte für Parameter
param_txt   equ      81h          ;ab hier Parameter-Text
dta_file    equ      9Eh          ;Beginn des Dateinamens in der DTA
buffer_lg   equ      40*1024      ;Länge des Puffers für Disk

;============ MACRO ========================================
print              macro  string              ;Bildschirmausgabe
                   lea    dx,string           ;Stringende = $
                   mov    ah,09h              ;DOS-Funktion 09
                   int    21h
endm

;************ CODE *****************************************
cseg              segment  'code'
                  assume   cs:cseg,ds:cseg,es:cseg,ss:cseg
                  org      100h

main:             mov    ah,30h               ;DOS-Version lesen
                  int    21h
                  cmp    ax,0A02h             ;DOS Version > 2.10?
                  jnb    ok_vers              ;ja
                  print no_vers               ;nein, --> Meldung
                  mov    ax,4C01h             ;und
                  int    21h                  ;Ende
```

Das Backup-Programm sichert komplette Verzeichnisse auch auf mehrere Disketten

```
no_param:     print err_msg_5
              mov   ax,4C01h          ;Fehler = 01
              int   21h

ok_vers:      print logo

param_check:  cld                     ;vorwärts
              mov   al,byte ptr cs:[param_txt]
              cmp   al,cr             ;wurden Parameter angegeben?
              je    no_param          ;nein --> Fehlermeldung

lwa_test:     mov   ax, word ptr cs:[param_txt+1]
                                      ;2. und 3. Zeichen
                                      ;(erstes = Blank)
              cmp   ax,':a'           ;Laufwerk a:?
              je    lwa_test2         ;ja
              cmp   ax,':A'           ;Laufwerk A:?
              je    lwa_test2         ;ja
              cmp   ah,':'            ;überhaupt LW angegeben?
              je    lwa_nein          ;ja, also nicht A und nicht
                                      ;default
              mov   ah,19h            ;nein --> default Laufwerk
              int   21h               ;feststellen
              cmp   al,0              ;Laufwerk A?
              jne   lwa_nein          ;nein, --> weitermachen
lwa_test2:    print err_msg_2         ;ja -> LW ändern?
              mov   ah,0              ;Tastenwert (scan code)
              int   16h               ;holen
              cmp   ah,24h            ;Taste J (Tasten Nr. 36)
              je    lw_chg2           ;ja
              mov   ax,4C01h          ;nein, --> Ende
              int   21h
lw_chg2:      mov   target_lw,'B'     ;Laufwerk ändern
              mov   default_lw,'B'

lwa_nein:     mov   bl,byte ptr cs:[param_lg]
              add   bl,param_txt
              mov   bh,0
              mov   byte ptr cs:[bx],0 ;CR auf 0
test:         dec   bl                ;Pfadnamen lesen und
                                      ;übertragen
              cmp   byte ptr cs:[bx],'\'
              je    found
              cmp   byte ptr cs:[bx],':'
              je    found
              cmp   byte ptr cs:[bx],' '
              je    default
              jmp   test
default:      mov   bx,offset path
              mov   word ptr cs:[inp_file],bx
              jmp   first_file

found:        mov   si,offset param_txt+1
              lea   di,path           ;Zieladresse
              sub   bx,param_txt      ;Zeichenzahl ausrechnen
              mov   cx,bx             ;Anzahl Zeichen übergeben
copy:         movsb                   ;und übertragen
              loop  copy
              mov   word ptr cs:[inp_file],di   ;Offset speichern

first_file:   call  find_first        ;ersten Eintrag suchen
              call  disk_chg          ;Diskette wechseln ....
```

```
transfer_fname:
              mov   si,offset dta_file        ;Startadresse
              mov   di,word ptr cs:[inp_file] ;Zieladresse
transfer_1:   lodsb
              stosb
              cmp   al,0              ;letztes Zeichen?
              je    trans_next        ;ja, --> für Zieldatei
              jmp   transfer_1        ;nein --> weiter
trans_next:   mov   si,offset dta_file
              lea   di,out_file       ;Zieladresse
transfer_2:   lodsb
              stosb
              cmp   al,0              ;letztes Zeichen?
              je    display_file      ;ja
              jmp   transfer_2        ;nein --> weiter

display_file: mov   bx,offset path    ;Pointer auf Quellendatei
              mov   ah,02             ;und ausgeben
abcd:         mov   dl,[bx]           ;mit Großbuchstaben
              cmp   dl,97             ;kleiner als a ?
              jb    abcd_end          ;ja
              cmp   dl,123            ;größer als z ?
              jae   abcd_end          ;ja
              and   dl,0DFh           ;nein --> Kleinbuchstabe ->
                                      ;wandeln
abcd_end:     cmp   dl,0              ;Ende?
              je    kopieren          ;ja
              int   21h               ;nein, -> ausgeben
              inc   bx
              jmp   abcd              ;weiter

kopieren:     call  open_source       ;Quellendatei öffnen
              jb    ende_error        ;Fehler aufgetreten
              call  open_target       ;Zieldatei erstellen
kopier_loop:  call  read_disk         ;Quellendatei lesen
              jb    ende_error        ;wenn Fehler
              cmp   ax,0              ;fertig?
              je    kopi_next         ;ja, nächste Datei
              call  write_disk        ;nein, -> Ziel
              jnc   kopier_loop       ;CF=0 -> kein Fehler
              cmp   ax,0FFh           ;Diskette voll ?
              jne   ende_error        ;nein, Fehler
              cmp   file_long,1       ;bereits einmal versucht?
              je    not_kopie         ;ja!
              mov   file_long,1       ;nein, --> Flag setzen
              call  disk_chg          ;neue Diskette ....
              jmp   kopieren          ;und nochmal versuchen

not_kopie:    print to_big            ;Hinweis ausgeben
              mov   file_long,0       ;Flag zurücksetzen
              jmp   next_match        ;nächste Datei

kopi_next:    mov   file_long,0       ;Datei war nicht zulang
              mov   bx,filenr_in      ;von Quellendatei
              mov   ax,5700h          ;Datum lesen
              int   21h
              mov   bx,filenr_out     ;und in Zieldatei
              mov   ax,5701h          ;Datum schreiben
              int   21h
kopi_error:   mov   bx,filenr_in      ;Quellendatei
              call  close_disk        ;schließen
              mov   bx,filenr_out     ;Zieldatei
              call  close_disk        ;schließen
```

```
;               call  kill_file         ;Quellendatei löschen
                                        ;nur bei MOVEKOMP !!
next_match:     print crlf
                clc
                mov   ah,4Fh            ;find next
                int   21h
                jc    ende              ;kein Eintrag mehr
                jmp   transfer_fname    ;Schleife

ende:           mov   ax,4C00h          ;Ende ohne Fehler
                int   21h

ende_error:     mov   bx,filenr_in      ;Quellendatei schließen
                call  close_disk
                mov   ax,4C01h          ;Ende mit Fehler
                int   21h

;*********************** Routinen ********************************

;=================================================================
;erste übereinstimmende Datei suchen und in DTA übergeben
;cx = Attribut
;Fehler: 2, 18
;=================================================================
find_first:     mov   cx,00100001b       ;Dateien Attribut =
                                         ;Archiv + Read/Only
                mov   dx,param_txt+1     ;Zeiger auf Pfadname
                mov   ah,4Eh             ;find first
                int   21h
                jc    search_error
                ret

search_error:   cmp   ax,02              ;Fehler 2? (Datei nicht
                                         ;gefunden)
                je    err2               ;ja
                print err_msg_3          ;nein, --> Fehler 18
                                         ;(keine weiteren Dateien)
                mov   4C01h              ;Ende mit Fehler
                int   21h
err2:           print dsk_err_2
                mov   4C01h              ;Ende mit Fehler
                int   21h

;=================================================================
;zum Diskettenwechsel auffordern, Bestätigung abwarten
;=================================================================
disk_chg:       print new_disk
                mov   ah,0
                int   16h
                cmp   ah,24h
                jne   disk_chg           ;Schleife bis J Taste
disk_chg1:      print crlflf
                ret

;=================================================================
;neue Datei eröffnen, bestehende wird überschrieben
;mögliche Fehler: 3, 4, 5
;Fehler 5 = Directory voll, oder R/O-Datei mit gl. Namen!
;=================================================================
open_target:    lea   dx,target_lw      ;Dateinamen Zieldatei
                mov   ah,3Ch            ;XENIX neue Datei
                mov   cx,20h            ;Attribut
                int   21h
```

```
                jc    inhalt_voll       ;CF=1 -> Fehler
                mov   filenr_out,ax     ;Dateinummer speichern
target_ok:      ret

inhalt_voll:    cmp   ax,5              ;Inhaltsverzeichnis voll ?
                jne'  error_dsk_o       ;nein
                print err_msg_1         ;ja
                call  disk_chg          ;Diskette wechseln!
                jmp   open_target       ;jetzt Datei eröffnen

;=================================================================
;bestehende Datei für Lesen öffnen
;mögliche Fehler: 2, 4, 5 und 12
;=================================================================
open_source:    lea   dx,path           ;Name Quellendatei
                mov   ax,3D00h          ;XENIX
                int   21h
                jc    error_dsk_o       ;CF=1 -> Fehler
                mov   filenr_in,ax      ;Dateinummer speichern
                ret

;=================================================================
;Fehlermeldung Diskette
;=================================================================
error_dsk_o:       cmp   ax,2
                   je    error_1        ;Fehlernr. 2
                   cmp   ax,3
                   je    error_1        ;Fehlernr. 3
                   cmp   ax,4
                   je    error_3        ;Fehlernr. 4

error_2:           print dsk_err_5      ;Fehlernr. 5,6,12
                   jmp   out
error_1:           print dsk_err_2
                   jmp   out
error_3:           print dsk_err_4
out:               stc                  ;CF=1 --> Fehler
                   ret

;=================================================================
;Datei lesen.   BX = Dateinummer,   AX = Anzahl gelesener Bytes
;mögliche Fehler: 5, 6
;=================================================================
read_disk:         mov   bx,filenr_in    ;Dateinummer
                   mov   cx,buffer_lg    ;Pufferlänge
                   lea   dx,buffer_dsk   ;Input Puffer
                   mov   ah,3Fh          ;XENIX "lesen"
                   int   21h
                   jae   dsk_read_ok     ;CF=0 -> kein Fehler
                   print dsk_err_5       ;Lesefehler
                   call  close_disk
                   stc                   ;CF=1 --> Fehler
                   ret
dsk_read_ok:       mov   cx,ax           ;gelesen Bytes übergeben
                   mov   bytes,ax        ;
                   ret

;=================================================================
;Datei schließen.       BX = Dateinummer
;mögliche Fehler: 6
;=================================================================
close_disk:        mov    ah,3Eh         ;XENIX
                   int    21h
```

```
                jnc     close_ok        ;CF=0 -> kein Fehler
                print   dsk_err_6
                stc                     ;CF=1 --> Fehler
close_ok:       ret

;=================================================================
;Datei schreiben.     BX = Dateinummer,   CX  = Anzahl Bytes,
;AX = geschriebene Bytes
;mögliche Fehler: 5, 6
;=================================================================
write_disk:     mov     bx,filenr_out   ;Dateinummer Zieldatei
                mov     cx,bytes        ;Anzahl Bytes
                lea     dx,buffer_dsk   ;Ausgabepuffer
                mov     ah,40h          ;XENIX
                int     21h
                jnc     write_ok        ;CF=0  -> kein Fehler
                print   dsk_err_5       ;Schreibfehler
                mov     ax,5
                stc                     ;Fehler --> CF=1
                ret
write_ok:       cmp     ax,bytes        ;alle Bytes geschrieben?
                jne     diskvoll        ;nein
                clc                     ;kein Fehler --> CF=0
                ret                     ;ja
diskvoll:       mov     bx,filenr_out
                call    close_disk      ;Zieldatei schließen
                lea     dx,target_lw    ;und
                call    delete_disk     ;löschen
                mov     bx,filenr_in    ;Quellendatei
                call    close_disk      ;schließen
                mov     ax,0FFh         ;Kennung Diskette voll!
                stc                     ;Fehler --> CF=1
                ret

file_long       db      0               ;0=ok, 1=wiederholter Versuch

;=================================================================
;Datei im Inhaltsverzeichnis löschen
;mögliche Fehler: 2, 5
;=================================================================
delete_disk:    mov     ah,41h          ;XENIX löschen
                int     21h
                jnc     delete_ok       ;CF = 0
                cmp     ax,02           ;Fehler Nr. 2?
                je      del_err1        ;ja
kill_error:     print dsk_err_5         ;dann Fehler 5
delete_ok:      ret
del_err1:       print dsk_err_2
                ret

;=================================================================
;Datei löschen, wenn Read/Only --> Attribut in Archive ändern,
;dann löschen
;           DIESE ROUTINE NUR BEI MOVEKOMP.COM erforderlich!
;Fehler: 2, 5  dann 3, 5
;=================================================================
;kill_file:     lea     dx,path         ;Quellendatei
;               mov     ah,41h          ;XENIX löschen
;               int     21h
;               jnc     delete_ok       ;kein Fehler
;               mov     cx,20h          ;R/O in Archive ändern
;               mov     ax,4301h        ;XENIX Dateimodus ändern
;               int     21h             ;
;               jc      kill_error      ;Fehler
;               jmp     kill_file       ;und nochmal

;****************************** DATEN ******************************
logo            db      cr,lf
                db      'COPYKOMP'      ;LOGO für COPYKOMP
;               db      'MOVEKOMP'      ;LOGO für MOVEKOMP
                db      '.COM   Ver. 1.10  (C) Copyright by'
                db      ' R. Kandlbinder  Feb. 1987',cr,lf
                db      'kopiert Dateien ganzer Verzeichnise '
                db      'auf mehrere Disketten im LW A: (B:)'
;               db      cr,lf,'und lösche die kopierten Dateien im '
;               db      'Verzeichnis' ;(beide Zeilen nur für MOVEKOMP)
                db      cr,lf,'$'
no_vers         db      cr,lf,'falsche DOS-Version (≥ 2.10 '
                db      'erforderlich)$'
err_msg_1       db      cr,lf,'Inhaltsverzeichnis voll deshalb:$'
new_disk        db      cr,lf,'neue formatierte Diskette in Laufwerk '
default_lw      db      'A: einlegen. Bereit (J)?$'
err_msg_2       db      cr,lf,'Laufwerk A: ist für '
                db      'COPYKOMP '
;               db      'MOVEKOMP '
                db      'das Ziellaufwerk!',cr,lf
                db      'Möchten Sie als Ziellaufwerk B: verwenden '
                db      '(J/N)?$'
err_msg_3       db      'keinen Pfad oder '
err_msg_4       db      'keinen passenden Eintrag gefunden',cr,lf,'$'
err_msg_5       db      cr,lf,'Parameterangaben fehlen:',cr,lf
                db      'Syntax: '
                db      'COPYKOMP '
;               db      'MOVEKOMP '
                db      '[d:][Pfad]Name.Erw',cr,lf,'z.B.  '
                db      'COPYKOMP '
;               db      'MOVEKOMP '
                db      'c:\dos\*.com',cr,lf,'$'
crlflf          db      lf
crlf            db      cr,lf,'$'
to_big          db      7,9,'Datei für Zieldiskette zu groß! Wird '
                db      'ausgelassen$'

dsk_err_2       db      cr,lf,'Datei nicht gefunden$'
dsk_err_3       db      cr,lf,'Pfad nicht gefunden$'
dsk_err_4       db      cr,lf,'zuviele Dateien geöffnet$'
dsk_err_5       db      cr,lf,'Datei verweigert Zugriff$'
dsk_err_6       db      cr,lf,'ungültige Behandlung$'
dsk_err_12      db      cr,lf,'ungültiger Zugriffscode$'

path            db      127 dup (?)     ;max. Länge der Parameter =
                                        ; 127 - Programmname
inp_file        dw      0               ;Offset
target_lw       db      'A:\'           ;Ziellaufwerk = A
out_file        db      14 dup (0)      ;Dateiname für LW A
filenr_out      dw      0               ;Nummer Zieldatei
filenr_in       dw      0               ;Nummer Quellendatei
bytes           dw      0               ;Anzahl der Bytes

buffer_dsk      equ     this byte       ;für Assembler

cseg            ends
                end     main
```

a) bei einer Pfadangabe den Backslash (\),
b) wurde nur ein Laufwerk angegeben, den Doppelpunkt,
c) wird das gültige Laufwerk und der gültige Pfad verwendet, ein Leerzeichen.

Wird ein Backslash oder Doppelpunkt gefunden, verzweigt das Programm zum Label *found*, und der Parametertext wird bis zum Dateinamen zum Label *path* übertragen. Wo nun der Dateiname beginnen würde, wird die Übertragung beendet und der Wert des DI-Registers gespeichert *(inp_file)*, damit später der jeweils gefundene Dateiname hier eingetragen werden kann.

Beispiel:
Übergebene Parameter: d:\cad\drawing-\projekt1*.dwg
In path steht: d:\cad\drawing\projekt1\
Ist vor dem Dateinamen ein Leerzeichen – also das gültige Laufwerk und der gültige Pfad – bedarf es keiner Textübertragung nach *path*. Denn der Dateiname beginnt ja sofort. Es muß somit nur der Offset gespeichert werden, dies geschieht ab Label *default*.
Beim Label first_file wird der erste übereinstimmende Eintrag gesucht. Hierzu wird die Unix-Funktion „find first" (4E) verwendet. In CX muß hierzu das Datei-Attribut stehen. Da COPYKOMP wie das DOS-Kommando COPY funktionieren soll, werden die beiden Attribute *Archive* und *Read/Only* verwendet. DS:DX muß auf eine ASCII-Zeichenfolge zeigen, die mit einem Null-Byte abgeschlossen ist. Die Parameter dafür sind bereits aufbereitet *(param_txt+1)*. Als Fehler kann nur auftreten: Pfad oder Datei nicht gefunden. Ist dies der Fall, wird das Programm beendet.
Ansonsten wird die DTA wie folgt aufgefüllt:
1 Byte – gefundenes Attribut
2 Bytes – Uhrzeit der Dateierstellung
2 Bytes – Datum der Dateierstellung
2 Bytes – niederwertiges Wort der Dateigröße
2 Bytes – höherwertiges Wort der Dateigröße
13 Bytes – Name und Erweiterung der gefundenen Datei, gefolgt von einem Null-Byte.

Im Code-Segment steht also ab Offset $9E_h$ der erste gefundene Dateiname. Mit der Routine *disk_chg* wird zum Diskettenwechsel *(new_disk)* aufgefordert und zur Bestätigung die Tastatur abgefragt. Hierbei ist nur die Taste „36" interessant. Dies ist die Taste „j", unabhängig davon, ob die Shift-, Alt- oder Control-Taste mitbetätigt wurde. Nach dem Diskettenwechsel wird beim Label *transfer_fname* der Dateiname aus der DTA an das String-Ende in *path*, also für die Quelldatei, sowie nach *out_file*, für die Zieldatei, übertragen.
Nun folgt die Bildschirmausgabe der zu kopierenden Datei (Label *display_file*). Die Ausgabe erfolgt dabei in Großbuchstaben. Anschließend wird die Quelldatei zum Lesen eröffnet und die Zieldatei mit dem Attribut Archive erstellt. Die Routine *open_source* kommt bei einem Fehler mit gesetztem Carry-Flag zurück. Bei dieser Anwendung können die Fehler 2, 4, 5 und 12 nicht auftreten, aus Prinzip ist trotzdem eine entsprechende Verzweigung nach *ende_error* vorgesehen.
Bei der Routine *open_target* kann eigentlich nur der Fehler 5 auftreten. Hierfür gibt es zwei Ursachen. Entweder ist das Inhaltsverzeichnis der Diskette voll oder eine Datei mit gleichem Namen und dem Attribut R/O besteht bereits. Auch hier ist aus Prinzip die Fehlerbehandlung bei einer solchen R/O-Datei in der Routine mit vorgesehen. Diese fordert zum Diskettenwechsel (err_msg_1 + Routine *disk_chg*) auf. Anschließend wird ein erneuter Versuch durchgeführt. Nun wird die Quelldatei in den Puffer gelesen. Anschließend wird der Puffer in die Zieldatei geschrieben. Dies geschieht so lange, bis das Ende der Quelldatei erreicht ist; dies ist der Fall, wenn null Byte gelesen wurden (AX=O) oder ein Schreibfehler auftrat (CF gesetzt). Ein Schreibfehler ist unter den gegebenen Umständen nur dann möglich, wenn die Diskette voll ist. Von der Routine *write_disk* wurde dabei bereits die unvollständige Zieldatei gelöscht, die Quelldatei geschlossen und in AX=FF übergeben. Um festzustellen, ob dies der zweite Versuch war, wird das Flag *file_long* abgefragt. Wenn es gesetzt ist, war dies der zweite Versuch, und die Datei ist für eine Diskette zu lang. Nach der entsprechenden Meldung *(to_big)* wird die nächste Datei kopiert. War es der erste Versuch, wird der Flag *file_long* gesetzt und zum Diskettenwechsel aufgefordert. Anschließend folgt ein neuer Kopierversuch.
Beim Label *kopi_next* wird das Flag *file_long* gelöscht und das Datum der Quelldatei auf die Zieldatei übertragen. Dann werden beide Dateien geschlossen. Nun wird zuerst CR, LF auf dem Bildschirm ausgegeben und dann der nächste Eintrag gesucht und kopiert. Findet sich kein Eintrag mehr, sind alle Dateien mit den vorgegebenen Attributen kopiert, und das Programm endet normal bei *ende*. Zur DOS-Ebene kehrt es mit dem Registerwert AL=00 zurück; bei Fehlern ist der Wert von AL=01.
Wer die Dateien anschließend sowieso löscht, kann sich diese „Arbeit" ersparen. Hierzu ist nur folgendes erforderlich: aktivieren der Zeile *call kill_file* im Label *kopi_error*, aktivieren der Routine *kill_file* und ändern des Namens im Logo, in err_msg_2 und err_msg_5 auf MOVEKOMP. Nun wird die Quelldatei nach dem erfolgreichen Kopieren automatisch gelöscht. Im Inhaltsverzeichnis stehen also nur noch Dateien, die zu groß waren und deshalb nicht kopiert wurden.
Das Label *kill_file* löscht mit der Unix-Funktion 41_h (nicht jedoch geschützte Dateien, „Read/Only"). Bei einer R/O-Datei ist das Carry-Flag gesetzt, und die Fehlermeldung lautet „Datei verweigert Zugriff". In diesem Fall ändert die Funktion 43_h das Attribut der Datei und löscht sie anschließend.

Literatur

[1] Betriebssystem DOS, technisches Handbuch. IBM.
[2] Programmierhandbuch MS-DOS 3.1. Microsoft.

Spruch des Monats

„Benchmarks do not lie. Liars do benchmarks."

C. H. Ting, Mitentwickler des Novix-Boards (Forth-Chip) im zugehörigen Handbuch „Footsteps In An Empty Valley"

Dr. H. Herre, Prof. H. Völz

Was ist berechenbar?

Das Problem der Berechenbarkeit zählt zu den Grundlagen der Rechentechnik-Informatik. Es betrifft die Frage, welche Probleme sich algorithmisch – das heißt: durch Computer-Programme – lösen lassen. Hier sollen dazu verschiedene Klassen zahlentheoretischer Funktionen definiert werden. Dabei gelangt man schließlich zu den allgemein rekursiven Funktionen, die am Beispiel der Ackermannschen Funktion untersucht werden.

Es seien genannt:

GDF gleichungsdefinierte Funktionen,
IBF iterativ berechenbare Funktionen,
EDF elementar definierbare Funktionen,
PRF primitiv rekursive Funktionen,
ARF allgemein rekursive Funktionen.
Zur Vereinfachung setzen wir voraus, daß alle Funktionen auf folgenden Grundoperationen aufbauen:

+ Addition,
– Subtraktion,
* Multiplikation;

der zulässige Zahlenbereich sei GZ und enthalte die ganzen Zahlen
$\ldots -3, -2, -1, 0, 1, 2, 3, \ldots$

Eine in dem Bereich GZ gegebene Funktion heiße berechenbar, wenn sich ein konstruktives Verfahren angeben läßt, daß nach Eingabe eines Arguments nach endlich vielen Schritten abbricht und den Wert der Funktion an dieser Stelle liefert. Offensichtlich ist jede programmierbare Funktion berechenbar. Die umgekehrte Behauptung, daß jede berechenbare Funktion auch programmierbar ist, ist der wesentliche Inhalt der CHURCHschen Hypothese [1].

Gleichungsdefinierte Funktionen

Die gleichungsdefinierten Funktionen (GDF) sind allgemein bekannt. Hierzu gehören beispielsweise:

$y = x * x,$
$z = x + y,$
$z = x^2 + 18 * x * y^3$ usw.

In der normalen Schreibweise steht also links vom Gleichheitszeichen das Ergebnis der Funktion und rechts der explizite Berechnungsweg. Werden k Eingangsgrößen zugelassen, so gilt allgemein:

```
INPUT X1, X2, ..., Xk
Y = EXPR (X1, ..., Xk)
PRINT Y
```

Hierbei ist EXPR (X1, ..., Xk) ein arithmetischer Ausdruck, der neben den Variablen X1, ..., Xk noch die Grundoperationen +, *, – und Zahlenkonstanten enthalten kann.

Etwas vereinfachend gesagt, entsprechen diese Funktionen einem Taschenrechnerniveau und bilden die Grundlage für die Mehrzahl der ingenieursmäßigen Berechnungen.

Mit Iteration berechenbare Funktionen

Bereits in relativ einfachen Fällen kann man keine explizite Gleichung für das Ergebnis angeben. Ein Musterbeispiel hierfür ist die Fakultät gemäß:

$f(x) = 1 * 2 * 3 * \ldots * x.$

Diese läßt sich durch das folgende Basic-Programm berechnen:

```
INPUT X
Y = 1
FOR I = 1 TO X
Y = Y * I
NEXT
PRINT Y
```

Dieses Programm ist zugleich ein Spezialfall eines iterativen bzw. eines LOOP-Programms. Nach [2] dürfen in einem LOOP-Programm nur folgende Anweisungen enthalten sein:

– Standardarithmetik,
– FOR-TO-NEXT,
– IF-THEN.

Eine IBF liegt nun genau dann vor, wenn für sie ein LOOP-Programm existiert. Dieses Programm stellt zugleich einen Lösungsalgorithmus dar.
Andere IBF sind:

SUM (X)	= die Summe der Zahlen 1, 2, 3, ..., X;
$\binom{n}{m}$	= Binomialkoeffizient;
s (X)	= Anzahl der Teiler von X;
p (X)	= Anzahl der Primzahlen bis X;
ggT (X,Y)	= größter gemeinsamer Teiler von X und Y.

Insbesondere aus den letzten drei Beispielen ist ersichtlich, daß zuweilen eine inhaltliche, nicht-algorithmische Beschreibung einer Funktion leichter anzugeben ist, als der dazugehörige Algorithmus. Es ergeben sich daher folgende Fragen: Läßt sich jede nicht-algorithmisch beschriebene Funktion durch einen Algorithmus repräsentieren, also programmieren? Wie läßt sich aus einer nicht-algorithmischen Beschreibung einer programmierbaren Funktion der Algorithmus gewinnen? Wie lassen sich die programmierbaren Funktionen klassifizieren?

Elementar definierbare Funktionen

Um die oben genannten Fragen zu beantworten, ist eine präzise Beschreibungssprache für Funktionen notwendig. Als besonders geeignet hierzu erweist sich der Prädikatenkalkül der 1. Stufe (PK 1). Neben den Individuenvariablen, die sich nach unserer Vereinbarung auf die ganzen Zahlen beziehen, können in den Ausdrücken dieses Kalküls $\exists$ als Existenz- und A als Allquantor sowie Boolesche Funktoren wie $\neg$ (not), $\wedge$ (and), $\vee$ (or), => (wenn, dann), <=> (ist äquivalent) verwendet werden. Beispiele solcher Definitionen sind:
– für das Maximum aus X und Y gilt:

$MAX(X,Y) = Z \Leftrightarrow (Z = Y \wedge X \leq Y) \vee (Z = X \wedge Y \leq X);$

– wenn Y die Zahl X teilt:

$DIV(X,Y) \Leftrightarrow \exists U (Y * U = X);$

– wenn X eine Quadratzahl sein soll:

QUAD (X) <=> ∃ Y (Y^2 = X).

Unschwer ist auch eine Definition für den größten gemeinsamen Teiler anzugeben:

ggT (X,Y) = Z <=> DIV (X,Z) ∧ DIV (Y,Z) ∧ ∀ U ((DIV (X,U) ∧ DIV (Y,U)) => Z ≥ u).

Derartige Definitionen heißen elementar-explizit. Sie sind dadurch gekennzeichnet, daß auf der rechten Seite der Definitionsgleichung eine Formel des PK 1 steht. In ihr tritt der zu definierende auf der linken Seite stehende Begriff nicht auf. Sie darf nur die bekannten Grundfunktionen und Grundrelationen enthalten.
Jede allgemeine Formel H(X1, ..., Xk) des PK 1 definiert (legt fest) eine Relation zwischen ganzen Zahlen. Jeder derartigen Formel H ist eine Funktion f_H zuordnenbar, die die Werte 0 und 1 annimmt:

$$f_H(N1, ..., Nk) \begin{cases} = 1, \text{ falls das k-Tupel (N1, ..., Nk) der ganzen Zahlen (Ni mit i = 1 bis k) die Formel H(X1, ..., Xk) wahr macht;} \\ = 0, \text{ andernfalls.} \end{cases}$$

f_H heißt die charakteristische Funktion der Formel H(X1, ..., Xk); die Funktion f_H ist selbst wieder elementar-explizit definierbar; z. B. läßt sich die Funktion f_{DIV} (X,Y) folgendermaßen definieren:

f_{DIV} (X,Y) = Z <=> Z = 1 ∧ ∃ U (X ∗ U = Y)) ∨ (Z = 0 ∧ ∀ U (X ∗ U ≠ Y)).

Mit der Verwendung des PK 1 erhalten so die oben gestellten Fragen einen präzisen Sinn.

Primitiv-rekursive Funktionen

Oft ist eine Funktion relativ leicht zu beschreiben, wenn sich ihre wesentlichen Eigenschaften durch allgemeine Gleichungen erfassen lassen. Ein Beispiel ist wieder die Fakultät:

FAK (0) = 1,
FAK (1) = 1,
FAK (X + 1) = FAK (X) ∗ (X + 1).

Die Funktion wird so also durch ein System von Gleichungen implizit oder axiomatisch erfaßt; im Unterschied zu den gleichungsdefinierten Funktionen kann hier die zu definierende Funktion auch auf der rechten Seite der Gleichungen auftreten. Das Beispiel von FAK läßt sich zu folgendem Grundschema verallgemeinern:

Es sei von zwei Funktionen der Argumente $X_1, X_2, ..., X_n, X1_{n+1}, X_{n+2}$ also g $(X_1, ..., X_n)$ und h $(X_1, ..., X_{n+2})$ ausgegangen. Dann läßt sich das folgende allgemeine Gleichungsschema für die Funktion f aufstellen:

f $(X_1, ..., X_n, 0)$ = g $(X_1, ..., X_n)$,
f $(X_1, ..., X_n, Y + 1)$ = h $(X_1, ..., X_n, Y, f$ $(X_1, ..., X_n, Y))$.

Es läßt sich zeigen, daß hierfür immer und eindeutig eine solche Funktion existiert. Aus einem Berechnungsverfahren für g und h läßt sich sogar immer eins für f konstruieren. Funktionen die sich mit derartigen Schemata und den Grundoperationen erfassen lassen, heißen primitiv rekursiv.

Folgerungen

Aus der mathematischen Logik läßt sich ableiten, daß die Klassen IBF und PRF gleich sind. Es kann sogar ein Computerprogramm entwickelt werden, daß jede primitiv rekursive Definition in ein äquivalentes iteratives Programm überführt und umgekehrt. Ein Fortran IV-Programm enthält [4]. Damit läßt sich auch ein Basic-Programm schreiben, was dieses leistet. Diese Aufgabe dürfte aber wahrscheinlich nicht ganz einfach sein. Allgemein interessant ist der Begriff der im intuitiven Sinne berechenbaren Funktionen BF. Es gilt offensichtlich:

GDF ∪ IBF ∪ PRF ⊆ BF. Man könnte nun annehmen, daß PRF = BF gilt. Alle „gewöhnlichen" zahlentheoretischen Funktionen lassen sich nämlich durch Iteration berechnen. Die mathematische Präzisierung der Klasse BF war eine wichtige Grundlage für die Untersuchungen von D. Hilbert im Rahmen seines Programms einer allgemeinen Beweistheorie und Metamathematik.

Innerhalb seiner berühmten 23 Probleme stellt er 1900 mit dem 10. Problem die Frage nach einer allgemeinen Methode für die Lösbarkeit von diophantischen (also ganzzahligen) Gleichungen. Hierzu ist ein präziser Algorithmenbegriff notwendig. 1926 fragte daher Hilbert, ob jede intuitiv berechenbare Funktion durch ein Schema der primitiven Rekursion beschreibbar ist. 1928 gab W. Ackermann in [3] eine berechenbare Funktion an, die nicht primitiv rekursiv berechenbar ist. Diese Funktion ist damit nicht durch ein LOOP-Programm programmierbar, so daß (in Standard-Basic) der GOTO-Befehl zusätzlich benötigt wird.

Die Ackermann-Funktion

Die folgende Variante der Ackermannschen Funktion aus [3] enthält zwei Variablen und ist durch das folgende Gleichungssystem definiert:

AK (0, Y) = Y + 1
AK (X, 0) = AK (X − 1, 1), falls X > 0
AK (X, Y) = AK (X − 1, AK (X, Y − 1))
falls X > 0, Y > 0.

Gegenüber den PRF fällt in der 3. Definitionszeile „nur" auf, daß in dem Rekursionsschema innerhalb der Funktion AK nochmal die Funktion AK steht. Die Funktion verweist also nicht einfach nur auf sich selbst, sondern in dem Selbstverweis ist noch einmal die Funktion implizit enthalten.

Zu jedem Paar (X, Y) von Argumenten läßt sich für AK mittels des obigen Gleichungssystems eine Folge von Termen (Ausdrücken) konstruieren. Im ersten Schritt erhalten wir einen AK-Term, deren erstes Argument um 1 reduziert ist und deren zweites Argument wiederum ein AK-Term ist, dessen zweites Argument um 1 reduziert ist. Genau dieses zweite Argument ließe sich nun so darstellen:

AK (X, Y − 1) =
AK (X − 1, AK (X, Y − 2))

und so ließe sich eine Folge ineinander verschachtelter AK-Terme konstruieren, die genau dann abbricht, wenn das zweite Argument des innersten AK-Terms gleich 0 ist. Auf diesen innersten AK-Term läßt sich dann die zweite Gleichung anwenden.
Damit entsteht ein AK-Term, in welchem die auftretenden Zahlenkonstanten kleiner als X bzw. Y sind. Dieser AK-Term läßt sich nach dem schon beschriebenen Schema der Schachtelung soweit reduzieren, daß schließlich ein AK-Term folgt, der nur noch die Zahlenkonstante 0 enthält. Dann kann die erste Gleichung angewendet werden und so entsteht schließlich das Ergebnis der Funktion AK für die Argumente X, Y.
Diese Beschreibung zeigt, daß die Funktion AK, wenn auch in komplizierter Schachtelung, berechenbar ist.
Zur Veranschaulichung des Verhaltens

```
10 REM: ## ACKERMANN ##
20 T=1: S=1: DIM A(1000): INPUT A(0),A(1)
30 IF T=0 THEN PRINT:PRINT"Ergebnis =";A(0): END
40 IF A(T-1)=0 THEN A(T-1)=A(T)+1: T=T-1: GOTO 80
50 IF A(T)=0 THEN A(T-1)=A(T-1)-1: A(T)=1: GOTO 80
60 A(T+1)=A(T)-1:A(T)=A(T-1):A(T-1)=A(T-1)-1:T=T+1
70 IF S<T THEN S=T
80 FOR X=0 TO T:PRINT A(X);: NEXT: PRINT: GOTO 30
READY

RUN
? 2,2
 1  2  1
 1  1  2  0
 1  1  1  1
 1  1  0  1  0
 1  1  0  0  1
 1  1  0  2
 1  1  3
 1  0  1  2
 1  0  0  1  1
 1  0  0  0  1  0
 1  0  0  0  0  1
 1  0  0  0  2
 1  0  0  3
 1  0  4
 1  5
 0  1  4
 0  0  1  3
 0. 0  0  1  2
 0  0  0  0  1  1
 0  0  0  0  0  1  0
 0  0  0  0  0  0  1
 0  0  0  0  0  .2
 0  0  0  0  3
 0  0  0  4
 0  0  5
 0  6
 7

Ergebnis = 7
READY
```

Bild 1. Die Ackermann-Funktion in Basic. Es werden der Kellerspeicher für die Tiefe der Rekursion und das Ergebnis ausgedruckt

der Schachtelung diene das Basic-Programm in *Bild 1*. Es benutzt ein Feld der Tiefe 1000 und legt hierin die entsprechenden verschachtelten Größen ab. Mit der Zeile 80 werden die verschachtelten Variablen ausgedruckt:

1 2 1 3 1

bedeutet dann also den Zustand

AK (1, AK (2, AK (1, AK (3, 1)))).

Am Beispiel AK (2,2) ist der Verlauf ausgedruckt und relativ gut zu verfolgen.
Die folgende Tabelle enthält einige Ergebnisse AK (I, J)

I/J	0	1	2	3	4	5	6
0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	5	7	9	11	13	15
3	5	13	29	61	125	253	509

Die Ergebnisse sind zugleich die Obergrenze zu- und abnehmender Schachtelungstiefe. Damit geben sie einen Hinweis auf die Größe der Rechenzeiten bei höheren, insbesondere bzgl. I gewählten Argumente. Für nicht in der Tabelle enthaltene Werte läßt sich abschätzen, daß folgende Beziehungen gelten:

$AK\ (4,\ 1) > 2^{14}$
$AK\ (4,\ 2) > 2^{2^{14}}$

$$AK\ (5,\ 1) > 2^{\cdot^{\cdot^{\cdot^{2^{14}}}}} \Big\} \quad 2^{14} - \text{mal}$$

Nicht unmittelbar einzusehen ist eine spezielle Eigenschaft der AK-Funktion. Hält man den ersten Parameter der Akkermann-Funktion fest, so entstehen 1-stellige Funktionen. Sie sind primitiv-rekursiv. Speziell gilt:

$AK_1\ (X) = AK\ (1,\ X) = 2 + X,$
$AK_2\ (X) = AK\ (2,\ X) = 3 + 2 * X.$

Diese Funktionen gehören zur Klasse GDF.
Bereits für die Funktionen $AK_3\ (X) = AK\ (3,\ X)$ ist eine Iteration notwendig, es gilt:

$AK_3\ (X) \geq 2^X * 5.$

Das LOOP-Programm für die Funktion AK_4 (X) hat bereits eine Verschachtelungstiefe von wenigstens 2. Für die Funktion $AK_5(X)$ ist es schon ziemlich schwierig, ein exaktes LOOP-Programm aufzuschreiben; die $AK_i(X)$ für große i sind nicht mehr komprehensibel. Der Leser kann sich von der Schwierigkeit dieser Aufgabe überzeugen, indem er ein Loop-Programm für $AK_5(X)$ aufzuschreiben versucht.
Man sieht also, daß sehr schnell alle Grenzen überschritten werden. Aber im Prinzip lassen sich selbst mit der beschränkten Arithmetik üblicher Rechner in einem gewissen Bereich noch Werte gewinnen. Genau wie bei der Arbeit [2] ist so wieder die Grenze erreicht, wo aus allgemeinen Funktionen nur ein kleiner Bereich zu berechnen ist. Darüber hinaus wird deutlich, daß es nicht zweckmäßig ist, jede primitiv-rekursive Funktion durch ein Loop-Programm zu programmieren. Andererseits liefert das Schema der primitiven und allgemeinen Rekursion Ausdrucksmittel, die das Wesentliche einer Funktion verständlicher erfassen.

Ergänzende Aussagen zu der Klasse EDF

Nach dem vorherigen Abschnitt ist die Klasse der allgemein rekursiven Funktionen umfassender, als die Klasse der primitiv-rekursiven Funktionen. Die Klasse der allgemein-rekursiven Funktionen läßt sich durch bestimmte Gleichungssysteme erfassen, wie es der Spezialfall der Ackermann-Funktion zeigt. Viele höhere Programmiersprachen besitzen Konzepte, die es gestatten, allgemeine Rekursivität zu definieren. In Lisp sind es zum Beispiel die rekursiven Funktionsdefinitionen, in Prolog rekursive Klauseln und in Algol (Pascal) rekursive Prozeduren.
Nach der These von CHURCH ist eine (überall definierte) zahlentheoretische Funktion berechenbar genau dann, wenn sie zu ARF gehört. Wenn man diese Hypothese akzeptiert, lassen sich die berechenbaren Funktionen durch die Klasse ARF präzisieren. Von jeder Programmiersprache wird vorausgesetzt, daß alle berechenbaren Funktionen in ihr programmierbar sind. Für Standard-Basic ist dies ebenfalls erfüllt, dabei ist jedoch der GOTO-Befehl unverzichtbar. Abschließend wollen wir die Beziehungen zwischen ARF und EDF untersuchen.
Aus der mathematischen Logik läßt sich ableiten, daß die Klasse ARF in EDF enthalten ist, d. h. jede allgemein-rekursive Funktion läßt sich durch eine explizite Definition des PK 1 erfassen. Es erhebt sich nun die Frage, ob jede zu EDF

```
10 CLS: DIM A(100): PRINT TAB(1Q);"## VACK ##": PRINT
20 DEF FN A(X)=X+1: DEF FN B(X)=X*Y
30 FOR I=1 TO 10: PRINT: PRINT;"I ="I
40  FOR J=1 TO 10: T=1: A(0)=I: A(1)=J
50   IF T=0 THEN PRINT A(0),:NEXT:NEXT:END
60 IF A(T-1)=0 THEN A(T-1)=FN A(A(T)):T=T-1: GOTO 50
70 Y=A(T-1): A(T-1)=Y-1: A(T+1)=FN B(A(T)): A(T)=Y-1: T=T+1 :GOTO 50
```

Bild 2. Ein Vorschlag zur Bildung von Ackermann-ähnlichen Funktionen

```
I = 1
 3            4            5
 6            7            8
 9            10           11
 12
I = 2
 6            8            10
 12           14           16
 18           20           22
 24
I = 3
 24           36           48
 60           72           84
 96           108          120
 132
I = 4
 732          1308         1884
 2460         3036         3612
 4188         4764         5340
 5916
I = 5
 1.74889E+06  3.40777E+06  5.06665E+06
 6.72553E+06  8.38441E+06  1.00433E+07
 1.17022E+07  1.33611E+07  1.50199E+07
 1.66788E+07
I = 6
 1.66606E+13  3.31719E+13  4.96832E+13
 6.61945E+13  8.27058E+13  9.92171E+13
 1.15728E+14  1.3224E+14   1.48751E+14
 1.65262E+14
I = 7
 1.91083E+27  3.81919E+27  5.72755E+27
 7.63591E+27  9.54428E+27  1.14526E+28
 1.3361E+28   1.52694E+28  1.71777E+28
 1.90861E+28
I = 8
```

Bild 3. Ein Lauf nach Bild 2

gehörige Funktion auch zu ARF gehört. Diese Frage hat eine unerwartete Antwort: Es gibt bereits ziemlich einfache Formeln, deren charakteristische Funktion nicht programmierbar ist. Diese mit der negativen Lösung des 10. Hilbertschen Problems zusammenhängende Frage soll in einer weiteren Arbeit untersucht werden.

Ackermann – ähnliche Funktionen

Die Ackermann-Funktion ist in ihrer Rekursion so gestaltet, daß die Argumente der weiter innen stehenden Terme stets nur kleinere Werte als die weiter außen stehende Argumente besitzen. Aus diesem Grund ist es schwierig, die Ackermann-Funktion zu modifizieren. Das folgende Rekursionsschema besitzt einerseits noch eine gewisse Ähnlichkeit mit dem Ackermann-Schema und läßt andererseits eine Vielzahl von rekursiven Funktionen über die beiden gleichungsdefinierten Funktionen

FN A (X) und FN B (X, Y)

zu. Die Gleichungen für das Rekursivschema lauten:

V (Ø, X) = FN A (X)
V (X, Y) = V (X−1, V (X−1, FN B (X, Y)))

```
4                 2
3  3  8
3  2  2  24
3  2  1  1  48
3  2  1  0  0  48
3  2  1  0  49
3  2  1  50
3  2  0  0  50
3  2  0  51
3  2  52
3  1  1  104
3  1  0  0  104
3  1  0  105
3  1  106
3  0  0  106
3  0  107
3  108
2  2  324
2  1  1  648
2  1  0  0  648
2  1  0  649
2  1  650
2  0  0  650
2  0  651
2  652
1  1  1304
1  0  0  1304
1  0  1305
1  1306
0  0  1306
0  1307
1308
```

Bild 4. Der Stack-Verlauf für A (4, 2)

Das zugehörige Basic-Programm mit zwei speziellen Funktionen zeigt *Bild 2*. Hierin sind die beiden Funktionen

FN A (X) = X + 1
und
FN B (X, Y) = X ∗ Y

verwendet. Das *Bild 3* zeigt die Ergebnisse der Funktion für V (X, Y) mit Ø < X < 8 und Ø < Y < 11.

Bild 4 listet den „Stack"-Verlauf für A (4, 2) auf. Andere Möglichkeiten für die beiden Funktionen sind z. B.

X + 2; 3 ∗ X bzw. Y ∗ Y, Y + X, Y + 1

auf diese Weise ist ein Experimentieren mit dieser veränderten Funktion möglich.

Literatur

[1] *Church, A.*: An unsolvable Problem of elementary number theory. Amer. J. Math. 58 (1936), 345–363.
[2] *Herre, H.; Völz, H.*: Goto und Standard-BASIC. mc 5/87.
[3] *Ackermann, W.*: Zum Hilbertschen Aufbau der reellen Zahlen. Math. Annalen 99 (1928), 118–133.
[4] *Rice, H.-G.*: Comm. Assoc. Computing Machinery, Vol. 8, N. 2, p. 114, 1965.

Neues Betriebssystem von Digital Research

Anfang Juni präsentierte Digital Research erstmals das Betriebssystem FlexOS 386, das sich eindeutig dem industriellen Markt und hier speziell dem CIM-Sektor zuwendet. Computer mit dem Intel-Prozessor 80386 werden mit FlexOS 386 zu einem flexiblen Echtzeit-Betriebsumfeld, dessen Zielgruppe ein von Digital Research als „Flexible Automation" charakterisierter Markt bildet. Zu den typischen Anwendungsgebieten gehören Steuerungen für Fertigungsanlagen, Laboratorien und die Verarbeitung finanzieller Transaktionen.
Mit FlexOS 386 verfügt der Anwender über ein Multitasking-/Multiuser-Betriebssystem, das mit verschiedenen Bildschirmfenstern arbeitet und den Parallelbetrieb völlig unterschiedlicher Software erlaubt, etwa von DOS- und Grafikprogrammen auf 8088/8086-Basis, von 80286-Programmen sowie von FlexOS-Software für den Protected-Mode des 80386.
Das Betriebssystem FlexOS 386 zeichnet sich besonders durch sehr kurze Interrupt-Latenzzeiten und schnellen Kontextwechsel aus. Das DOS-Application-Environment unterstützt GEM und DOS 3.3 und eignet sich für PC-DOS-Applikationen wie z. B. Lotus 1–2–3, Multimate und dBase III Plus.
Das OEM-Lizenzprodukt FlexOS 386 soll nach Angaben von Digital Research ab September ausgeliefert werden. Derzeit gibt es ein Systems Builder Kit inklusive einer Tagesschulung für 4275 DM. Mit dem betriebsfertigen System, das einen High-C-Compiler von Metaware und das GEM Virtual Device Interface zur Anbindung an Grafiksysteme enthält, können Applikationen für das FlexOS 386-Betriebssystem auf allen 100 %ig kompatiblen ATs entwickelt werden. Zur Ausstattung des Systems Builder Kits gehören des weiteren Muster-Treiber und Systemmodule zur Implementierung von FlexOS-Systemen.
Bereits Anfang des Jahres hatte Digital Research FlexOS 286 für den Intel-80286 vorgestellt. Für Zellencontroller, die auf dem 80186 basieren, gibt es das Betriebssystem FlexOS 186.
Nach Unterlagen von Digital Research

Ingrid Trommer

Einführung in Unix

Teil 4: Shell als Programmiersprache

Die Shell ist eine eigenständige Programmiersprache. Sie kennt die meisten Konstrukte einer höheren Programmiersprache. Shell-Prozeduren machen Unix zu einem leistungsfähigen Betriebssystem. Unsere Autorin schildert, wie Shell-Prozeduren erstellt und getestet werden.

Alle Befehle, auch solche die sich über mehrere Zeilen erstrecken, kann der Anwender interaktiv am Terminal eingeben. Da sich viele Befehlsfolgen in der Praxis wiederholen, ist es besser, Shell-Prozeduren zu verwenden.

Um eine Shell-Prozedur zu erstellen, gibt man mit einem Editor eine Befehlsfolge ein und speichert diese im System ab. Mit dem Befehl „chmod" (change mode) teilt man dem System mit, daß die abgespeicherte Datei ausführbaren Code enthält. Durch die Eingabe „chmod +x meinprog" wird die vorher editierte Datei meinprog ausführbar.
Die Eingabe „meinprog" ruft diese Shell-Prozedur auf, vorausgesetzt das aktuelle Directory ist im Suchpfad enthalten.
Bild 1 zeigt, wie der Inhalt der Datei „meinprog" am Terminal ausgegeben wird. Anschließend wird „meinprog" als ausführbar gekennzeichnet und aufgerufen. Die Befehlsfolge in dieser Datei stellt die Anzahl der momentan eingeloggten Benutzer fest. In sämtlichen Beispielen sind die Eingaben des Anwenders unterstrichen dargestellt, um sie von den Ausgaben des Rechners zu unterscheiden.

```
$ cat meinprog
who | wc -l
$ chmod +x meinprog
$ meinprog
      4
$
```

Bild 1. Eine einfache Shell-Prozedur wird erstellt und aufgerufen

```
if (logischer Ausdruck)
then
.
.
else
.
.
fi
```

Bild 2. Syntax der if-Abfrage

```
if (logischer Ausdruck)
then
.
.
elif (logischer Ausdruck)
  then
  .
  .
fi
```

Bild 3. Syntax der if-Abfrage mit else-Klausel

Sprach-Syntax

Die Prozedur „meinprog" besteht aus einer Zeile. Eine Prozedur darf mehrere Zeilen enthalten, die dann nacheinander ausgeführt werden. Abhängig von bestimmten Bedingungen kann in jeder Programmiersprache dieser sequentielle Ablauf verlassen werden.

Welche Kontrollstrukturen kennt die Shell? Die einfachste Art einer Verzweigung stellt die if-Abfrage dar (*Bild 2*).
Die else-Klausel kann weggelassen werden. In der else-Klausel darf eine weitere if-Abfrage stehen (*Bild 3*).
Prozedurteile, die mit „if" anfangen, werden mit „fi" abgeschlossen. if-Konstrukte dürfen beliebig tief geschachtelt sein. Man sollte jedoch beachten, daß verschachtelte if-Anweisungen unübersichtlich sind. Es gibt auch Rechner, die bei zu großer Verschachtelungstiefe Probleme mit ihrem Stack bekommen. Unübersichtliche Shell-Prozeduren mit zahlreichen elif-Anweisungen (else if) können oft durch die case-Anweisung vereinfacht werden.

Tabelle 2: Mit diesen Primär-Operatoren dürfen die Operatoren aus Tabelle 1 verknüpft werden

Operator	Funktion
!	unäre Negation (nur ein Operand)
−a	binäres Und
−o	binäres Oder

Einen logischen Ausdruck, also wahr oder unwahr, erhält man mit dem test-Kommando. In *Tabelle 1* sind die Optionen des test-Kommandos zusammengefaßt. In *Tabelle 2* sehen Sie mit welchen Primär-Operatoren die Operatoren aus Tabelle 1 verknüpft werden dürfen.
Eine Shell-Prozedur, die u.a. prüft, ob eine Datei ein Directory enthält, wird in *Bild 4* vorgestellt. Ruft man „ifsh" mit dem positionalen Parameter „file1" auf, so wird beim Ablauf jedesmal „$1" durch „file1" ersetzt. Im ersten „if" wird getestet, ob „file1" vorhanden ist. Wenn ja, wird sie durch das Drucker-Spooler-Programm „lp" ausgedruckt. Das nächste „if" prüft, ob „file1" ein Dateiver-

Tabelle 1: Varianten des test-Kommandos

test	Option	Der logische Ausdruck ist wahr, wenn:
	−r datei	datei existiert und lesbar ist
	−w datei	datei existiert und beschrieben werden darf
	−x datei	datei existiert und ausführbar ist
	−f datei	datei existiert und eine normale Datei ist
	−d datei	datei existiert und ein Directory ist
	−s datei	datei existiert und der Inhalt ungleich Null ist
	−z s1	die Länge des Strings s1 Null ist
	−n s1	die Länge des Strings s1 nicht Null ist
	s1 = s2	String s2 identisch mit String s1 ist
	s1 != s2	String s2 nicht identisch mit String s1 ist
	n1 −eq n2	Integer n1 und n2 algebraisch gleich sind. Jeder der Vergleichsoperationen −ne, −ge, −lt, −le kann statt −eq verwendet werden

```
$ cat ifsh
  if test -f $1
  then
     lp $1
  elif test -d $1
    then
       echo $1 ist ein Directory
    else
       echo datei $1 nicht vorhanden
  fi
```

Bild 4. Beispiel einer Shell-Prozedur mit if-Abfragen und else-Klauseln

zeichnis ist. Wenn ja, wird eine entsprechende Meldung ausgegeben. Wenn beides nicht der Fall ist, wird auch dazu eine Meldung am Terminal ausgegeben.

Schleifen-Konstruktionen

Mit der for-Schleife kann eine Variable mit verschiedenen Werten besetzt, und die Schleife mit dem jeweils geänderten Variablen-Wert durchlaufen werden.

```
for var in wert1 .. wertn
do
.
.
.
done
```

Bild 5. Syntax der for-Schleife

```
$ cat shl
for i in ab cc dd
do
echo $i
done
$ shl
ab
cc
dd
$
```

Bild 6. Anwendung einer for-Schleife

```
while (logischer Ausdruck)
do
.
.
.
done
```

Bild 7. Syntax der while-Schleife

```
until (logischer Ausdruck)
do
.
.
done
```

Bild 8. Syntax der until-Schleife

```
case variable in
var1) .
      .
      ;;
var2) .
      .
      ;;
.
.
varx) .
      .
      ;;
*)    .
      .
      ;;
esac
```

Bild 9. Syntax der case-Anweisung

Bild 5 zeigt die for-Schleife in ihrer allgemeinen Form. Eine einfache Anwendung ist in *Bild 6* dargestellt.
In der while-Schleife wird eine Bedingung getestet und die Befehle in der Schleife so lange durchlaufen, bis der Test als Ergebnis „falsch" zurückliefert. In *Bild 7* ist die Syntax in einer while-Schleife dargestellt.
Im Gegensatz dazu führt eine until-Konstruktion die Befehle so lange aus, bis der Test ein wahr zurückgibt. *Bild 8* zeigt die allgemeine Form einer until-Konstruktion.
Die case-Anweisung führt einen mehrfachen Vergleich durch. Der Wert des Selektors (hier: der Wert der Shell-Variablen „variable") wird mit mehreren Marken verglichen. Das Ende einer case-Konstruktion wird mit „esac" gekennzeichnet. In *Bild 9* sehen Sie die Syntax dieser Anweisung.
Stimmt der Wert von „variable" z. B. mit var1 überein, werden die nachfolgenden Befehle, die durch einen Punkt angedeutet sind, bis zu den beiden Semikolons ausgeführt. Anschließend wird die case-Konstruktion verlassen. Der mit dem Asterisk (*) gekennzeichnete Fall ist eine Übereinstimmung, die immer zutrifft. Hier kann man Befehle unterbringen, die ausgeführt werden sollen, wenn keiner der vorher angegebenen Fälle zutrifft. Natürlich ist der Asterisk optional.
Wie einfach es ist, mit einer Shell-Prozedur ein Menü zu schreiben, sehen Sie in *Bild 10*. Das Doppelkreuz leitet einen Kommentar in Shell-Prozeduren ein. *Bild 11* zeigt, wie die Shell-Prozedur arbeitet.

Ersetzungs-Mechanismen

Das Beispiel in *Bild 12* erläutert, wie die vorbesetzten Variablen in einer Shell-Prozedur verwendet werden. Beim Aufruf erwartet die Prozedur mindestens zwei Übergabe-Parameter. Der erste Parameter ist eine Buchstabenkombination. Die folgenden Parameter sind Dateinamen. Die Dateinamen werden durch die Prozedur so geändert, daß die Buchstabenkombination getrennt durch einen Punkt an den Namen angehängt wird. In diesem Beispiel sind der shift-Befehl und die Syntax mit den geschweiften Klammern neu.
Der shift-Befehl verringert die Anzahl der Parameter (Symbol:$#) um eins. Außerdem geht von den Aufruf-Parametern der Positionsparameter „$1" verloren. Der alte Positionsparameter „$2" wird zum neuen „$1", das alte „$3" zum neuen „$2", usw. Mit diesem Mechanismus kann man innerhalb einer Prozedur alle übergebenen Parameter als Positionsparameter „$1" behandeln.

```
$ cat menush
while true                             #Endlosschleife true ist immer wahr
do
cat menudat                            #menudat enthält Text des Menüs
echo
echo "Auswahl 0 - 3 : \c"              #"\c": Cursor geht nicht auf nächste Zeile
read antwort                           #antwort wird Wert über Terminal zugewiesen
#Variable antwort ist damit definiert
case $antwort in                       #Wert von antwort wird mit $antwort abgefragt
0) exit
   ;;
1) date
   echo
   ;;
2) pwd
   echo
   ;;
3) echo "Welches Directory ?"
   read dir
   ls $dir
   echo
   ;;
*) echo "Falsche Eingabe"              #immer wenn Eingabe nicht zwischen 0 u. 3
   ;;
esac                                   #Ende von case
done                                   #Ende von while


$ cat menudat
Gib 0 fuer Ende
    1 fuer Datum und Uhrzeit
    2 fuer Directory-Name
    3 fuer Directory-Liste
```

Bild 10. Ein Menü ist mit einer Shell-Prozedur schnell programmiert

Außerdem lassen sich so mehr als neun positionale Parameter verarbeiten. Mit geschweiften Klammern kann man mit den Werten von Shell-Variablen durch Buchstaben-Kombinationen neue Namen erzeugen.
Um die Wirkung der Shell-Prozedur „paramsh" zu sehen, wird mit „ls" der Inhalt des momentanen Directories ausgegeben (*Bild 13*). Die Shell-Prozedur „paramsh" wird mit „dat new*" aufgerufen. Die Shell expandiert „new*" in alle Dateinamen, die mit „new" anfangen, einschließlich von „new" selbst. Der Prozedur „paramsh" werden die Namen „new newuu newxx newyy" übergeben. Diese Expandierung wird von der aufrufenden Shell vorgenommen. Die Shell-Prozedur weiß davon nichts.
Es gibt weitere Ersetzungsmechanismen in der Shell, die bis jetzt noch nicht besprochen wurden.
Die Ausgabe eines Befehls besetzt eine Variable, wenn man den Befehl in umgekehrte Apostroph-Zeichen setzt. Mit olddir= 'pwd' wird die Variable „olddir" mit dem Namen des momentanen

```
$ cat paramsh
if test $# -le 2                       #Anzahl der Argumente <=2 ?
then
   echo $0: usage $0 append file1 .. filen    #Fehlermeldung im Unix-Stil
fi                                     #Ende von if
echo "Anzahl Argumente $#"             #Anzahl der Argumente ausgeben
echo "Argumente $*"                    #die Argumente selbst ausgeben
append=$1                              #Var. append wird auf Wert von $1 gesetzt
shift                                  #$2 wird zu $1, $3 zu $2 usw.
echo "nach shift $#"                   #Anzahl nach shift um eins geringer
echo "Argumente nach shift $*"         #das ehemalige $1 gibt es nicht mehr
while test $# -ne 0                    #solange Anzahl der Argumente nicht 0
do
  file=$1                              #Var. file wird auf Wert von $1 gesetzt
  echo "Datei $file wird in ${file}.$append umbenannt"
  mv $file ${file}.${append}           #Umbenennen von file
  shift
done                                   #Ende von while
$
```

Bild 12. Diese Shell-Prozedur zeigt die Anwendung vorbesetzter Variablen und positionaler Parameter

```
$ ls
dat       menush    newsh     newxx     paramsh   testsh
menudat   new       newuu     newyy     rmdatei
$ paramsh dat new*
Anzahl Argumente 6
Argumente dat new newsh newuu newxx newyy
nach shift 5
Argumente nach shift new newsh newuu newxx newyy
Datei new wird in new.dat umbenannt
Datei newsh wird in newsh.dat umbenannt
Datei newuu wird in newuu.dat umbenannt
Datei newxx wird in newxx.dat umbenannt
Datei newyy wird in newyy.dat umbenannt
$
```

Bild 13. Eine Anwendung aus Bild 12 auf dem Bildschirm

```
$ cat rmdatei
new.dat
newuu.dat
newxx.dat
newyy.dat
$ ls
menudat    new.dat     newxx.dat   paramsh
menush     newuu.dat   newyy.dat   rmdatei
$ rm `cat rmdatei`
$ ls
menudat   menush   paramsh   rmdatei
$
```

Bild 14. Das Kommando zwischen den beiden umgekehrten Apostrophen wird durch dessen Ergebnis ersetzt

```
$ menush
Gib 0 fuer Ende
    1 fuer Datum und Uhrzeit
    2 fuer Directory-Name
    3 fuer Directory-Liste

Auswahl 0 - 3: 1
Mon Feb 16 17:23:20 GMT 1987

Gib 0 fuer Ende
    1 fuer Datum und Uhrzeit
    2 fuer Directory-Name
    3 fuer Directory-Liste

Auswahl 0 - 3 : 2
/usr/hugo/sh

Gib 0 fuer Ende
    1 fuer Datum und Uhrzeit
    2 fuer Directory-Name
    3 fuer Directory-Liste

Auswahl 0 - 3 : 3
Welches Directory ?
/usr/hugo
buch         daten1       daten5       sh          text.2zeil
cprog        daten2       oha          text        text.dup

Gib 0 fuer Ende
    1 fuer Datum und Uhrzeit
    2 fuer Directory-Name
    3 fuer Directory-Liste

Auswahl 0 - 3 : 6
Falsche Eingabe
Gib 0 fuer Ende
    1 fuer Datum und Uhrzeit
    2 fuer Directory-Name
    3 fuer Directory-Liste

Auswahl 0 - 3 : 0
$
```

Bild 11. So stellt sich das Menü aus Bild 10 auf dem Bildschirm dar

```
$ a=5
$ a=`expr $a + 1`
$ echo $a
6
```

Bild 15. Shell-Prozeduren können rechnen!

Dateiverzeichnisses besetzt. Auch Übergabeparameter kann man auf diese Art besetzen (*Bild 14*).

Rechnen mit der Shell

Auch Rechnen kann man innerhalb der Shell. Mit der in *Bild 15* gezeigten Befehlsfolge wird der Wert der Variablen „a" um eins erhöht. Unter dem Schlüsselwort „expr" sind z.B. die Operatoren „+", „–", „*"(Multiplikation), „/"(Division) und „%"(Modulo) möglich. Vorsicht bei Metazeichen! Die Substitution von Metazeichen läßt sich bei einem einzelnen Zeichen mit „\" verhindern. Will man bei einem längeren Ausdruck jede Ersetzung vermeiden, so erreicht man das durch Einschließen des Ausdrucks in Apostrophe. Bei einem Ausdruck, der in doppelte Apostrophe gesetzt ist, werden alle Shellvariablen ersetzt, jedoch keine Dateinamen und Metazeichen.

Eingebaute Shell-Kommandos

Damit das Unix-System einen Befehl findet, müssen zwei Bedingungen erfüllt sein:

1. Der Befehl muß als ausführbar gekennzeichnet sein.

2. Das Directory, in dem sich der Befehl befindet, muß in der Variablen „PATH" enthalten sein.

Ist der gleiche Name in mehreren Dateiverzeichnissen enthalten, so wird immer der Befehl gefunden, dessen Directory in der Zeichenkette der Variablen „PATH" weiter links steht. Man könnte der Ansicht sein, daß man bei gleichlautenden Befehlsnamen nur die eigenen Directories in der Variablen „PATH" nach vorne setzen muß, damit die eigenen Prozeduren ausgeführt werden. Dieses ist nur bedingt richtig. Die Shell enthält in sich eine ganze Reihe häufig gebrauchter Befehle, um diese schneller ausführen zu können. Für den Anwender bedeutet dies, daß er bestimmte Namen nie bei eigenen Prozeduren verwenden darf, da nicht über den PATH-Mechanismus auf sie zugegriffen wird. Für diese Befehle ist außerdem die Umleitung der Ein- oder Ausgabe verboten. Die Ausgabe dieser Befehle erfolgt immer über die Standard-Ausgabe. Einige dieser Bezeichnungen sind:
cd, echo, exit, export, pwd, read, set, shift, test.

Der Name „test" ist sehr beliebt bei Prozeduren, über deren Funktionsweise man sich nicht im klaren ist. Verwendet man diesen Namen, wird die Funktionsweise dieser Prozedur nie klar ersichtlich werden. Die Shell ruft das „built in" Kommando auf und gibt den nächsten Prompt aus, weil Parameter zu „test" fehlen.

Es würde zu weit führen alle Befehle, die zu dieser Gruppe gehören aufzuführen. Es sei hierzu auf das Unix-Referenz-Manual verwiesen. Zwei weitere Anweisungen dieser Gruppe sind „break" und „continue".

Mit „break" kann eine for-, while- oder until-Schleife verlassen werden. Mit „continue" geht man zum Schleifenende, bricht also die momentane Iteration ab.

```
$ cat ifsh
if test -f $1
then
  lp $1
elif test -d $1
  then
    echo Directory
  else echo $1 not found
fi
$ ll ifsh
-rw-rw-rw-   1 hugo      users           97 Feb 25 20:13 ifsh
$ sh -x ifsh testsh
+ test  -f testsh
+ lp testsh
request id is lp-250 (1 file)
$

$ sh -x ifsh ww
+ test -f ww
+ test -d ww
+ echo ww not found
ww not found
$
```

Bild 17. Aufruf einer Shell-Prozedur mit „–x"-Option

Testen von Shell-Prozeduren

Das bisher Gesagte zeigt, daß Shell- Prozeduren beliebig kompliziert sein können. Wie jedes Programm, führt auch eine Shell-Prozedur nur das aus, was man ihr sagt, aber nie das, was man meint. Da es nur sehr wenige Menschen gibt, bei denen ein komplexes Programm auf Anhieb läuft, haben die Unix-Entwickler dem Anwender auch Test-Möglichkeiten zur Verfügung gestellt.

Der einfachste Test wird mit dem echo-Befehl durchgeführt. Will man sichergehen, daß die Prozedur „paramsh" (*Bild 13*) die Dateinamen richtig umbenennt, so läßt man den Befehl „mv" erst einmal weg. Startet man den Prozeß zunächst ohne den mv-Befehl, so gibt er das Programm wegen des echo-Befehls auf dem Terminal aus und zeigt, wie die Dateien umbenannt werden. Sobald das Ergebnis den eigenen Vorstellungen entspricht, startet man den Prozeß mit dem mv-Befehl. Um der Unix-Philosophie zu entsprechen, daß keine Nachrichten gute Nachrichten sind, sollte man dann auch das „echo" entfernen.

Beim Aufruf von Prozeduren darf der Anwender Optionen setzen. Jede Prozedur wird automatisch durch eine Sub-Shell ausgeführt. Dies erreicht man auch durch eine explizite Eingabe des Befehls „sh". Ruft man eine Unter-Shell durch „sh" auf, so dürfen Befehls-Optionen mit angegeben werden. In diesem Fall ist es nicht notwendig, daß die Prozedur durch „chmod +x" als ausführbar gekennzeichnet wurde.

Eine dieser Optionen ist „-v" für „verbose". Damit gibt die Shell die Befehle aus, während sie ausgeführt werden. Diese Option ist besonders nützlich um Syntaxfehler zu finden. Zum Testen der Prozedur „ifsh" muß man folgenden Befehl eingeben:

$ sh -v ifsh

Um die Wirkung der Option zu verdeutlichen, wurde in „ifsh" ein Fehler eingebaut. Das „then" nach der elif-Zeile fehlt. Das Ergebnis sieht man in *Bild 16*. Eine weitere Option lautet -x für „execute". Benutzt man diese Option, wird jeder Befehl mit allen ersetzten Variablen ausgegeben. Hier wurde das fehlende „then" wieder eingesetzt. Das Beispiel in *Bild 17* zeigt die Wirkung der „-x"-Option an zwei verschiedenen Übergabeparametern, so daß zwei Wege der drei möglichen Verzweigungen durchlaufen werden.

```
$ cat ifsh
if test -f $1
then
  lp $1
elif test -d $1
    echo Directory
  else echo $1 not found
fi
$ ll ifsh
-rw-rw-rw-   1 hugo     users           90 Feb 25 20:11 ifsh
$ sh -v ifsh
if test -f $1
then
  lp $1
elif test -d $1
    echo Directory
  else ifsh: syntax error at line 6: `else' unexpected
$
```

Bild 16. Aufruf einer Shell-Prozedur mit „–v"-Option

Joachim Buhmann, Robert Divko, Helge Ritter, Klaus Schulten

Physik und Gehirn

Wie dynamische Modelle von Nervennetzen natürliche Intelligenz erklären

Nicht nur die Grundlagen zukünftiger Rechner-Generationen, auch die Erklärung des menschlichen Verstandes ist seit jeher Forschungsziel der künstlichen Intelligenz. Unsere Autoren, Mitglieder der Arbeitsgruppe Theoretische Biophysik an der Technischen Universität München, geben im folgenden Beitrag eine Einführung in diesen Bereich und liefern auch Programme für eigene Experimente.

Vor etwa dreißig Jahren begannen Mathematiker wie Shannon und Wiener, die damals neuentwickelten Rechenmaschinen für komplizierte Entscheidungsprobleme einzusetzen. In einer Phase großer Euphorie entstanden damals erste Programme, die z. B. Schach und Dame spielen konnten und dabei Fähigkeiten zeigten, die gemeinhin als intelligent angesehen werden. Aber trotz enormer weiterer Anstrengungen blieben die Erfolge weit hinter den Erwartungen der Pioniere zurück.

Die Fähigkeiten selbst einfacher Tiere, komplexe Sinneseindrücke – wie etwa Bilder auf der Netzhaut – in Sekundenbruchteilen erfassen, die ungeheure Datenfülle scheinbar mühelos auf die wesentlichen Informationen zu reduzieren und mit einer vernünftigen Reaktion des Bewegungsapparats (z. B. Flucht) zu beantworten, überschreitet weit die bisherigen Möglichkeiten künstlicher Gehirne. Computerprogramme benötigen für vergleichbare Aufgaben immer noch Minuten bis Stunden. Bei der Bilderkennung kommen sie über eine bescheidene Szenenanalyse nicht hinaus; im Falle der Bewegungssteuerung von Robotern gelingen ihnen nur stereotype und unbeholfene Bewegungen.

Worin also unterscheidet sich das biologische Gehirn von herkömmlichen (von-Neumann-)Computern und weshalb ist es so überlegen? Im Gegensatz zum von-Neumann-Rechner, der nur einen Zentralprozessor besitzt, einzelne Befehle eines Programmes daher nacheinander abarbeitet und dazu Daten aus einem getrennten Speicher liest bzw. dort abspeichert, gibt es im Gehirn weder die Beschränkung auf einen Prozessor noch die Trennung zwischen Zentraleinheit und Speicher.

Im Gehirn „arbeiten" mehr als 1 Milliarde Nervenzellen (Neuronen genannt) nebeneinander und fungieren dabei als Prozessoren und Speicherelemente zugleich. Über ein Netz aus etwa 10 Billionen Verbindungen (sogenannten Synapsen) teilen sie sich ihren aktuellen elektrischen Aktivitätszustand ständig gegenseitig mit. Jedes einzelne Neuron summiert ununterbrochen die an den Synapsen einlaufenden elektrischen Signale und reagiert seinerseits, falls das Summensignal einen bestimmten Schwellwert überschreitet, mit einem Signal an die anderen Nervenzellen. Diese Aktion stellt den einzigen „Prozessorbefehl" dar, über den das einzelne Neuron verfügt. Dennoch kann man damit jede beliebige Rechenoperation aufbauen, wie bereits 1943 McCulloch und Pitts gezeigt haben.

Aus dem gleichzeitigen Zusammenwirken der riesigen Anzahl solcher elementarer Operationen im Gehirn ergibt sich eine komplizierte Dynamik, durch die seine Rechenleistung zustande kommt. Der Ablauf dieser Dynamik wird von zwei Hauptfaktoren beeinflußt: Zum einen nehmen an ihr die von den Wahrnehmungssensoren gelieferten Signale teil, zum anderen wird sie durch die Plastizität synaptischer Verbindungen gesteuert. So werden beim Lernen systematisch auf mehreren Zeitskalen (Kurzzeitgedächtnis ... Langzeitgedächtnis) die Verknüpfungen zwischen den Neuronen geändert. Während die Neuronen also als Schwellwert-Elemente die Prozessoren des Gehirns darstellen, bilden ihre synaptischen Verknüpfungen das Speichermedium.

Die Physik hat sich seit langem mit dem Verhalten von Systemen aus sehr vielen miteinander wechselwirkenden Komponenten beschäftigt und dabei systematisch den Zusammenhang zwischen den Eigenschaften der Einzelkomponente und dem Verhalten des Gesamtsystems untersucht. Es stellte sich heraus, daß höchst einfache Einzelkomponenten und Wechselwirkungen äußerst vielfältige Ordnungszustände des Gesamtsystems als Resultat einer Dynamik, bei der Zufallsprozesse eine wichtige Rolle spielen, zustande bringen (man denke z. B. an Kristallisationsprozesse). Eine Reihe von Wissenschaftlern versucht heute, diese Ergebnisse der physikalischen Forschung auf neuronale Netzwerke zu übertragen und dadurch die Frage zu beantworten, welche grundlegenden Eigenschaften von Nervenzellen und ihren synaptischen Verbindungen (Wechselwirkungen) die Leistungen der natürlichen Intelligenz hervorbringen können.

Die Sichtweise von Nervennetzwerken als dynamische Systeme hat die Künstliche Intelligenz neu befruchtet und einen neuen Hoffnungsschimmer aufleuchten lassen: wenn sich Rechner und Programme am Vorbild neuronaler Netzwerke und ihrer Dynamik orientieren, könnten vielleicht einige hartnäckige Probleme der KI wesentlich besser und schneller als bisher gelöst werden. Dem Leser wird auffallen, daß wir hier vorsichtig formulieren; der weit verbreitete Optimismus soll aber nicht verschwiegen werden: während interessierte Wissenschaftler sich 1985 und 1986 noch im kleinen Kreis von 50 Teilnehmern in den Bergen von Utah trafen, mußte 1987 bereits ein Kongreß in San Diego abgehalten werden, zu dem sich 1400 Teilnehmer angesagt hatten.

Wir wollen im folgenden die Leistungen von dynamischen Netzwerkmodellen an Hand von drei Beispielen und drei Pascal-Programmen demonstrieren: ein Netzwerk aus Modellneuronen, das Stereobildpaare erkennt, ein dynamisches System, welches zwei Nervennetze miteinander verschaltet und ein neuronales Netzwerk, das als assoziativer Speicher für einfache Muster dient.

Stereosehen

Als erstes Beispiel betrachten wir ein Problem aus dem Bereich des Computer-Sehens. Während es für einen Menschen keinen bewußten Aufwand bedeutet, visuelle Information zu verarbeiten, haben selbst schnellste Rechner Schwierigkeiten, komplexe Bildinhalte zu erkennen. Für den Einsatz von Robotern in der modernen Fertigungstechnik ist jedoch eine schnelle Erfassung und Interpretation großer Mengen von Bilddaten bei erträglichen Kosten erforderlich. Die verwendeten Algorithmen müssen dazu eine Reduktion der Datenflut auf die für die Mustererkennung relevanten Bildmerkmale erreichen und mit diesen eine Beschreibung der Umwelt liefern. Dabei sollten kleine Fehler in den Aufnahmekameras oder Rauschquellen in Übertragungskanälen durch fehlertolerante Algorithmen ausgebessert werden.
Ein interessanter Teilaspekt des Computer-Sehens ist das Stereo-Problem, die Rekonstruktion räumlicher Information aus zwei von leicht unterschiedlichen Blickrichtungen aufgenommenen Bildern.
Der visuelle Cortex (ein Gehirnareal) des Menschen ist in der Lage, aus den leicht unterschiedlichen Bildern auf den Netzhäuten beider Augen in Bruchteilen von Sekunden einen räumlichen Eindruck der beobachteten Szene zu gewinnen. Bereits Anfang der 60er Jahre entdeckte Bela Julesz, daß das Gehirn auch imstande ist, Stereobilder, die keinerlei objektbezogene Information enthalten, räumlich zu interpretieren. Ein Beispiel ist das in *Bild 1* dargestellte Zufalls-Stereogramm. Der linke Teil in Bild 1 besteht aus zufällig mit gleicher Wahrscheinlichkeit schwarz oder weiß gewählten Bildpunkten. Der rechte Bildteil ist, bis auf einen um einige Punkte nach links (zur Nase hin) verschobenen Ausschnitt aus der Bildmitte, identisch zum linken Teil. Das so erzeugte Bildpaar vermittelt bei Betrachtung durch ein Stereoskop (ein umgedrehter Feldstecher ist ebenso geeignet) den Eindruck eines vor dem Hintergrund schwebenden Quadrats. Während identische Bilder im Stereoskop als ebene Fläche erscheinen, erscheint das schwebende Quadrat um so höher, je größer die Verschiebung ist. Da diese Zufallsmuster außer dieser Verschiebung – auch Disparität genannt – keinerlei Information enthalten, kann der Tiefeneindruck *allein* aus der Verschiebung zugehöriger Bildpunkte ermittelt werden.
Um nun die Tiefe verschiedener Bildteile zu berechnen, sind folgende vier Schritte nötig:

1. Wähle einen Punkt im linken Bild
2. Finde den entsprechenden Punkt im rechten Bild
3. Messe die Verschiebung der beiden Punkte
4. Berechne daraus die Entfernung

Das eigentliche Problem liegt dabei in der Zuordnung der Punkte beider Bilder. Kennt man nämlich diese Zuordnung, so lassen sich die Disparitäten einfach bestimmen; daraus die Entfernungen zu berechnen ist ein rein geometrisches Problem. Die entscheidende Frage beim Stereosehen ist also, korrespondierende Punkte im linken Bild und rechten Bildteil zu identifizieren. Dieses Problem wird gerade beim Zufalls-Stereogramm besonders deutlich. Betrachtet man etwa den weißen Punkt in der linken oberen Ecke im Teil von Bild 1, so könnte der zugehörige Punkt im rechten Bild ebenfalls in der linken oberen Ecke liegen, oder aber um einen Bildpunkt verschoben sein oder irgend einem anderen weißen Bildpunkt in derselben Reihe entsprechen! Der Ausweg aus der Vielzahl von Zuordnungsmöglichkeiten besteht darin, nur solche Korrespondenzen zuzulassen, die physikalisch sinnvollen Interpretationen des Bildes entsprechen. Zwei grundlegende Annahmen über die Umwelt gehen dabei ein:

Eindeutigkeit:
Ein Punkt auf der Oberfläche eines Körpers hat eine eindeutige Position im Raum.

Stetigkeit:
Die Oberfläche von Objekten ist im allgemeinen glatt (außer an Objektkanten).

Sinnvolle Zuordnungen zwischen den Bildpunkten im linken und rechten Teil von Bild 1 müßen also so erfolgen, daß jedem Bildpunkt im linken Teil genau einer im rechten Teil entspricht (Eindeutigkeit) und benachbarte Punkte des





Bild 1. Räumliches Sehen ist auch ohne objektbezogene Information möglich: betrachtet man dieses Zufallsmuster durch ein Stereoskop, so ergibt sich ein räumlicher Eindruck

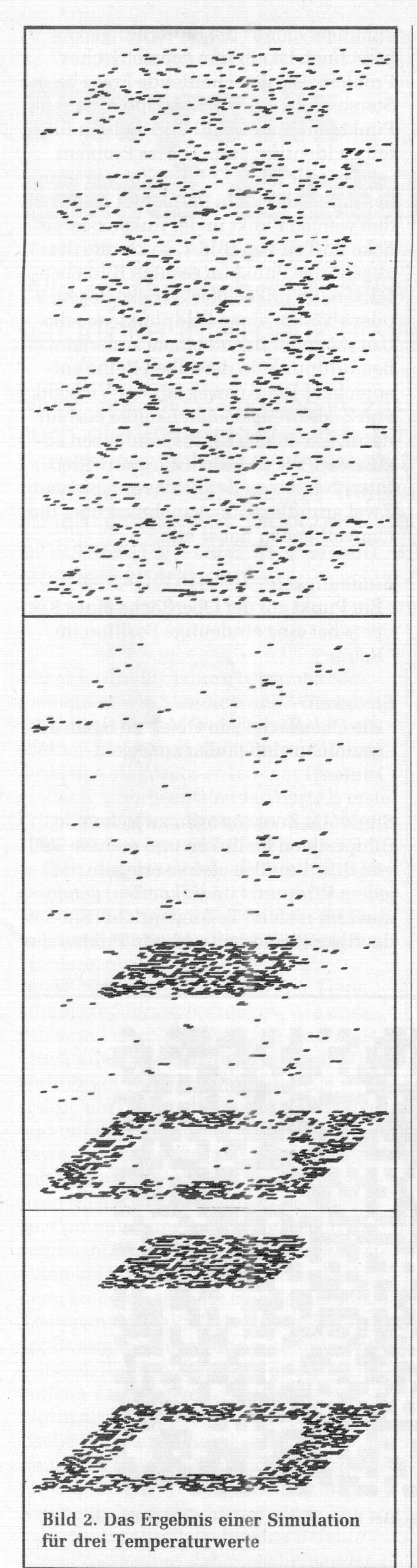


Bild 2. Das Ergebnis einer Simulation für drei Temperaturwerte

linken Bildes gegenüber entsprechenden Punkten im rechten Bild ungefähr gleich weit verschoben sind (Stetigkeit). Diese Annahmen sind im folgenden Algorithmus zur Lösung des Stereo-Problems berücksichtigt.
Für jeden Punkt (i,j) des linken Bildes kodiert die Richtung eines Zeigers S_{ij}, wie groß die Disparität des korrespondierenden Punktes im rechten Bild (und damit Entfernung des Beobachters zu dem betrachteten Objekt) ist. Um diejenigen Zeigerstellungen, welche den besten Interpretationen des Stereobildpaars entsprechen, zu bestimmen, ordnet man dem System der Zeiger eine Energie so zu, daß einer guten Bildinterpretation energetisch niedrige Zustände des Systems entsprechen. Benachbarte Zeiger liefern bei gleichen Werten der angezeigten Disparität einen energieabsenkenden Beitrag und erhalten dadurch die Tendenz, sich parallel auszurichten. Die beste Interpretation der Stereobilder entspricht dann der Konfiguration von Zeigern mit der niedrigsten Gesamtenergie. Die im Beispielprogramm verwendete Energie ist

Formel 1:

$$E = -W \sum_{<(i,j),(k,l)>} \delta(S_{ij}, S_{kl}) + \sum_{(i,j)} \left| lpic(i,j) - rpic(i - s_{ij}, j) \right|$$

$$\delta(a,b) = \begin{cases} 1 \text{ falls } a = b, \\ 0 \text{ sonst.} \end{cases}$$

Der erste Energieterm liefert einen absenkenden Beitrag, falls benachbarte Zeiger gleiche Werte S_{ij} und S_{kl} der Disparität anzeigen (die spitzen Klammern bedeuten dabei eine Summation über nächste Nachbarn).
Der zweite Term vergleicht die Intensitäten in den beiden Stereobildern 'lpic' und 'rpic' miteinander und liefert dann einen niedrigen Energiebeitrag, wenn ein Bildpunkt (i,j) im linken Bild die gleiche Farbe (schwarz oder weiß) wie der um S_{ij} verschobene Bildpunkt im rechten Bild hat. Dadurch wird sichergestellt, daß schwarze Pixel im linken Bild auch bevorzugt schwarzen Punkten im rechten Bild zugeordnet werden. Der Faktor 'W' legt das relative Gewicht zwischen beiden Energiebeiträgen fest.
Wie erhält man nun die Konfiguration von Zeigern mit der geringsten Gesamtenergie, die der besten Interpretation des Stereobildpaares entspricht? Dazu ist es hilfreich, eine Analogie zu Festkörpern herzustellen. Einen besonders regelmäßigen Kristall, der einem niedrigen energetischen Zustand des Festkörpers entspricht, erhält man durch Erhitzen und anschließend vorsichtiges Abkühlen des Materials.
Diesen Prozeß kann man auf dem Computer simulieren, indem man an den Zeigern Zufallskräfte angreifen läßt, die bei hoher Temperatur heftige Zufallsbewegungen verursachen. Erniedrigt man jedoch diese Zufallskräfte, so gewinnt der ordnende Einfluß der Energiefunktion mehr und mehr die Oberhand und das System nimmt im abgekühlten Zustand Zeigerstellungen an, die den in *Bild 2* dargestellten Bildtiefen entsprechen.
Der Abkühlvorgang muß dabei so langsam erfolgen, daß das System nicht zu schnell „ausfriert" und in einem energetischen Nebenminimum, das nicht der bestmöglichen Interpretation entspricht, stecken bleibt.
Wie kann man nun dieses System von Zeigern auf dem Rechner realisieren? Dazu betrachten wir das Beispielprogramm (*Bild 3*): Das Zeigersystem 's' und die beiden Stereobilder 'lpic' und 'rpic' sind in zweidimensionalen Feldern vom Typ 'picture' gespeichert. Nachdem in 'inputs' die Eingabeparameter eingelesen sind, werden die beiden Zufalls-Stereogramme wie oben beschrieben erzeugt und die Anfangsstellungen des Zeigersystems zufällig gewählt, wobei für jeden Bildpunkt Werte im Bereich 0...maxdisp möglich sind.
In der Hauptschleife wird dann das physikalische Verhalten der Zeiger für immer kleinere Temperaturen T simuliert. Der Einfluß der Temperatur besteht in einer zufälligen Fluktuation der Zeigerstellungen, die nach folgenden Regeln verändert werden (Metropolis Algorithmus): Ein Zeiger des Bildes wird als Kandidat für eine Änderung ausgesucht und ein neue Einstellung zufällig gewählt. Erniedrigt sich dadurch die Energie, so wird dieser neue Zeigerwert akzeptiert. Erhöht sich die Energie um einen Betrag ΔE, so wird die Änderung nur mit einer Wahrscheinlichkeit $\exp(-\Delta E / T)$ akzeptiert. Bei hoher Temperatur T wird mit $\exp(-\Delta E / T) \simeq 1$ fast jeder neue Wert akzeptiert, wogegen bei niedriger Temperatur durch $\exp(-\Delta E / T) \simeq 0$ die Wahrscheinlich-

keit, daß ein neuer Wert akzeptiert wird, klein ist. Die Energieänderung durch eine neue Zeigereinstellung wird in der Funktion 'calculate' berechnet und in der Prozedur 'anneal' wird dann entschieden, ob eine neue Zeigerstellung akzeptiertwird. Dieser Vorgang wird für jeden Punkt des Gitters mehrere Male wiederholt (nmcs ≃ 20 ist ein typischer Wert). Nach jedem Temperaturschritt wird dann mit der Prozedur 'print' das jeweils erhaltene Spinmuster ausgegeben.
Für kleine Temperatur (etwa $T < 0{,}02$) ist das Muster der Disparitäten stabil und entspricht einer optimalen Interpretation des Stereobildes durch das Spinsystem. Es zeigt bei unserem Beispielprogramm eine Verschiebung eines zentralen Quadrats um 3 Bildpunkte.
Die Energieänderung des Systems bei Änderung eines Spins hängt nur lokal von den Werten anderer Spins ab. Dies ermöglicht eine für jeden Punkt unabhängige Berechnung der Dynamik des Systems, wodurch es ideal für Parallelrechner ist.

Der Algorithmus für das Stereosehen kann erweitert werden, indem mehrere Bildmerkmale, wie etwa Farbe, Intensität, Geschwindigkeiten oder Kanten, in Zeigern kodiert werden. Damit erhöht sich die Leistungsfähigkeit des Systems, wodurch es möglich ist, natürliche Bilder zu verarbeiten. In der aktuellen Forschung werden damit Probleme wie die Rekonstruktion verrauschter Bilder, der Farbinvarianz beim Sehen oder das Stereo-Problem bei natürlichen Bildern untersucht.

Selbstorganisierende Abbildungen

Im vorstehenden Abschnitt haben wir ein Modell für die Funktionsweise eines „Gehirnmoduls" zum Stereosehen kennengelernt. Die meisten Verarbeitungsleistungen des Gehirns kommen aber nicht mit einem isolierten, einzelnen Modul aus, sondern sind auf die enge Zusammenarbeit mehrerer Module angewiesen.

Die einzelnen Module sind dabei als zweidimensionale Gebiete (sogenannte Rindenfelder) der Hirnrinde realisiert, die miteinander durch Bündel aus Nervenfasern, sogenannten „neuronalen Projektionen", verbunden sind. Diese Verbindungen unterliegen einem wichtigen Organisationsprinzip: sie sind nachbarschaftserhaltend, d. h., benachbarte Neuronen eines Rindenfelds A sind wieder mit benachbarten Neuronen eines Rindenfelds B verbunden. Dieselbe Eigenschaft der Erhaltung von Nachbarschaftsbeziehungen findet sich auch bei den Verbindungen zwischen den Sinneszellen vieler unserer Sinnesrezeptoren und den entsprechenden Rindenfeldern im Gehirn, in denen diese Sinnesreize weiterverarbeitet werden. So gibt es z. B. ein Rindenfeld im Gehirn, das in der geschilderten nachbarschaftserhaltenden Weise durch Nervenfasern mit den Tastsensoren in unserer Hautoberfläche verbunden ist. Wie werden solche hochgradig geordneten Verbindungen zwischen einzelnen Neuronenschichten erzeugt? Angesichts der Komplexität der herzustellenden Verbindungen scheidet bei den höheren Organismen eine detaillierte genetische Codierung des Schaltplanes aus (es ist aber vielleicht interessant zu wissen, daß die Natur diesen Weg bei kleinen Nervensystemen, z. B. von Insekten, noch beschritten hat). Man geht davon aus, daß die genetische Information nur eine grobe Vorverschaltung liefert, die sich dann in den Einzelheiten selbst organisiert und strukturiert. Eine attraktive Hypothese dazu ist, daß das Gehirn unter dem Einfluß von Sinnesreizen im Laufe der Zeit die anfänglichen Verbindungen verändert. Von der Nachahmung dieser Fähigkeit auf dem Computer erhoffen wir uns neue Einsichten in die Funktionsweise des Gehirns, aber erhoffen auch die Möglichkeit der Selbstorganisation eines Teils der Verbindungsstruktur in künftigen parallelen Rechnern zu erschließen.
Ein auf den finnischen Physiker Kohonen zurückgehendes Computermodell für einen geeigneten Verschaltungsalgorithmus wird an unserem Institut seit einiger Zeit untersucht und soll im folgenden am Beispiel der Herausbildung einer geordneten Verbindung zwischen den Rezeptoren einer Sinnesoberfläche, wie sie z. B. von unserer Haut mit ihren Tastrezeptoren gebildet wird, und den Neuronen eines Rindenfelds im Gehirn erklärt werden. In dem Computermodell wird das Rindenfeld als ein quadrati-

```
program stereosehen(input,output); (* Juli 87, Robert Divko *)
const
   n      = 20;              (* Dimension *)
   maxdisp = 5;              (* Maximale Disparitaet *)
   mnt = 20;                 (* Max Zahl v. Temperaturschritten *)
type picture = array [1..n,1..n] of integer;
var
   nt, nmcs, i, seed: integer;
   w, t, acceptance: real;
   tem: array [1..mnt] of real;
   s, lpic, rpic: picture;

function random (var i : integer) : real; (* Zufallszahlen aus [0,1] *)
begin
   i := (10009 * i + 1);
   random := 0.5 * ( 1 - i / maxint );
end;

procedure print(var pic: picture ); (* gibt Bild aus *)
var
   x, y: integer;
begin
   writeln;
   for y := 1 to n do
      for x := 1 to n do
      begin
            write(pic[x,y]:1);
            if x = n then writeln;
      end;
end;

procedure inputs;
var
   i: integer;
begin
   write('Wie viele Temperaturschritte? ');
   readln(nt);
   for i:= 1 to nt do
   begin
      write(i:1,'ter Temperaturschritt (von ',nt:1,'en) := ');
      readln(tem[i]);
   end;
   write('Wie oft durchs Gitter? ');
   readln(nmcs);
```

Bild 3. Mit diesem Programm kann das Zeiger-System für räumliches Sehen auf Rechnern simuliert werden ▶

```
   write('Wechselwirkung := ');
   readln(w);
end;

procedure start;
const
   xmin = 6;             (* Bildausschnitt links *)
   xmax = 16;            (* rechts *)
   ymin = 6;             (* unten *)
   ymax = 15;            (* oben *)
   disp = 3;             (* Verschiebung *)
   p    = 0.5;           (* Wahrscheinlichkeit fuer weiss *)
var
   x, y, i: integer;
   rand: real;
begin
   seed := 3297;                 (* Zufallszahlen initialisieren *)
   inputs;                       (* Eingabeparameter *)
   for x := 1 to n do
      for y := 1 to n do
      begin                      (* Stereobildpaar *)
         if(p >= random(seed)) then
            lpic[x,y] := 1       (* schwarz *)
         else
            lpic[x,y] := 0;      (*  weiss  *)
         rpic[x,y] := lpic[x,y]; (* linkes und rechtes Muster *)
      end;
   (* Verschiebe Rechteck um disp nach links *)
   for x := xmin to xmax do
      for y := ymin to ymax do
         rpic[x-disp,y] := lpic[x,y];
   (* fuelle entstandene Luecke mit Zufallsmuster *)
   for x := (xmax - disp + 1) to xmax do
      for y := ymin to ymax do
      begin
         if(p >= random(seed)) then rpic[x,y] := 1
         else lpic[x,y] := 0;
      end;
   (* Anfangsverteilung der Disparitszustaende *)
   for x := 1 to n do
      for y := 1 to n do
      begin
         rand := random(seed);
         s[x,y] := trunc( (maxdisp + 1) * rand);
      end;
end;

function nen(l, i, j:integer): integer;
(* Zahl der Nachbarn mit gleicher Disparitaet *)
var
   sum: integer;
begin
   sum :=0;
   if (i+1) <= n then if (s[i+1,j] = l) then sum := sum + 1;
   if (i-1) >=1  then if (s[i-1,j] = l) then sum := sum + 1;
   if (j+1) <=n  then if (s[i,j+1] = l) then sum := sum + 1;
   if (j-1) >=1  then if (s[i,j-1] = l) then sum := sum + 1;
   nen := sum;
end;

function calculate (disp, x, y: integer): real; (* berechne Energieaenderung *)
var
   match_new, match_old, match_diff: integer;
   ix, n_new, n_old, n_diff: integer;
begin
   (* Vergleiche durch disp einander zugeordnete Bildpunkte *)
   ix := x - disp;
   if ix < 1 then match_new := 1
   else match_new := abs ( lpic[x,y] - rpic[ix,y]);
   ix := x - s[x,y];
   if ix < 1 then match_new := 1
   else match_old := abs ( lpic[x,y] - rpic[x-s[x,y],y]);
   match_diff := match_new - match_old;
   (* vergleiche benachbarte Disparitaetszustaende *)
   n_new := nen(disp,x,y);
   n_old := nen(s[x,y],x,y);
   n_diff := n_new - n_old;
   calculate := - w * n_diff + match_diff;
end;

procedure anneal;
var
   de, crit, rand: real;
   i, x, y, new, accepted: integer;
begin
   accepted := 0;
   for i :=1 to nmcs do              (* nmcs mal durchs Gitter *)
      for x := 1 to n do
         for y := 1 to n do
         begin
            (* Neue Hypothese fuer Disparitaet *)
            rand := random(seed);
            new := s[x,y] + 1 + trunc (maxdisp * rand);
            if new > maxdisp then new := new - maxdisp-1;
            (* Bestimmung der Huepfwahrscheinlichkeit *)
            de := calculate( new, x, y);
            if (de <= 0.0) then crit := 1.0
            else crit := exp( - de /t );
            if(crit >= random(seed)) then (* akzepiert *)
            begin
               s[x,y] := new;
               accepted := accepted + 1;
            end;
         end;
   acceptance := accepted /nmcs /n /n;
end;

begin
   start;            (* Initialisierungen *)
   for i := 1 to nt do
   begin
      t := tem[i];
      anneal;        (* simuliere Spinsystem *)
      writeln;
      writeln('Temperatur :=',t:3:3);
      writeln('Akzeptanz :=',acceptance:3:3);
      print(s);      (* Ausgabe *)
   end;
end.
```

sches Gitter dargestellt. Jeder Gitterpunkt steht für ein Neuron, das durch seine beiden Gitterpunktkoordinaten (i,j) identifiziert wird. Jedes Neuron wird über eine Nervenfaser mit einem Punkt der Sinnesoberfläche verbunden. Im Programm geschieht dies dadurch, daß für jeden Gitterpunkt (i,j) ein 2-dimensionaler Vektor $\vec{w}_{ij}$ eingerichtet wird, dessen beide Komponenten w_{ij1} und w_{ij2} mit den Koordinaten x bzw. y des angeschlossenen Punktes auf der Sinnesoberfläche belegt werden (wir machen die Annahme, an jedem Punkt (x,y) einen Tastsensor treffen zu können und wählen als Sinnesoberfläche das Quadrat
$0 \leq x \leq 1, 0 \leq y \leq 1$).

Als Anfangszustand wählt das Programm eine völlig zufällige Zuordnung zwischen Neuronen (i,j) und Orten (x,y) (Prozedur 'initialize'). Diese Zuordnung wird graphisch dargestellt, indem für jedes der „Neuronen" (i,j) des Gitters der Ort des Quadrats, mit der es gemäß der durch $\vec{w}_{ij}$ gegebenen Zuordnung verbunden ist, durch einen Punkt markiert wird, wobei die Punkte zu im Gitter benachbart liegenden Neuronen durch eine Linie verbunden werden (Prozedur 'draw').
Die anfänglich völlig ungeordnete Zuordnung zwischen Neuronen (= Gitterpunkten) und Rezeptorpunkten (x,y) soll sich allein aufgrund der Einwirkung einer Folge von Tastreizen auf der Sinnesoberfläche so ordnen, daß am Ende benachbarte Rezeptorpunkte mit benachbarten Neuronen im Gitter verbunden und die Verbindungen gleichmässig über die Sinnesoberfläche verteilt sind. Zur Lösung dieser Aufgabe müssen die Neuronen die in der Folge der Tastreize nur implizit enthaltene Information über die Form der Sinnesoberfläche entdekken und zugleich untereinander so kooperieren, daß sich am Ende eine Zuordnung mit den gewünschten Eigenschaften ergibt. Dies wird durch folgenden Algorithmus ermöglicht:

1. Jedesmal, wenn ein neuer Tastreiz – etwa an der Stelle (x,y) – auftrifft, wird dasjenige Neuron (i_0,j_0) ermittelt, das für die Stelle (x,y) „am zuständigsten" ist, d.h. für das der Abstand zwischen dem Vektor $w_{i_0j_0}$ und $(x,y)^T$ am kleinsten ist (Prozedur 'bestmatch').

2. Für alle Neuronen (i,j) in der Nachbarschaft von (i_0,j_0) (einschließlich (i_0,j_0) selbst) werden die Verbindungen w_{ij} in Richtung auf (x,y) hin verschoben. Dabei nimmt das Ausmaß der Verschiebung mit wachsender Entfernung von (i_0,j_0) und zunehmender Iterationszeit t ab. Dies wird durch die Anwendung der Vorschrift

Formel 2:

$$\vec{w}_{ij} \rightarrow \vec{w}_{ij} + H\left[(i-i_0)^2 + (j-j_0)^2, t\right] \cdot ((x,y)^T - \vec{w}_{ij})$$

für jedes Neuron (i,j) erreicht (Prozedur 'adapt'). Dabei ist
$H(D,t) = H_0 * \exp(-a_0 t - D/b^2(t))$ und
$b(t) = b_0 * \exp(-c_0 t)$.

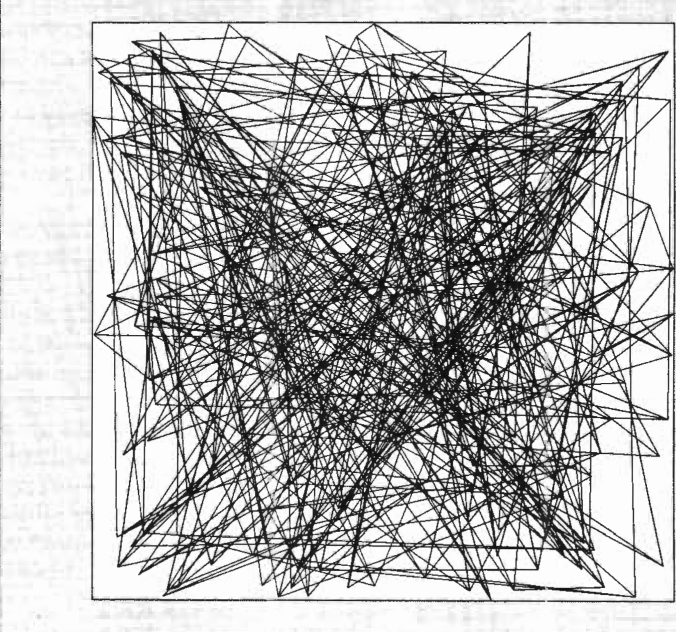
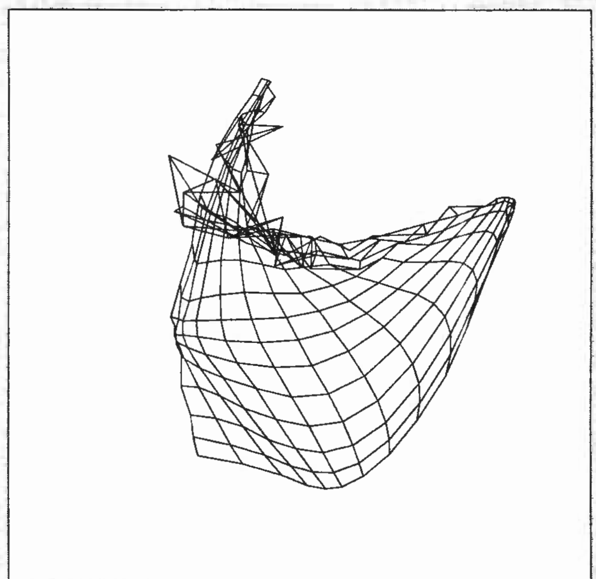
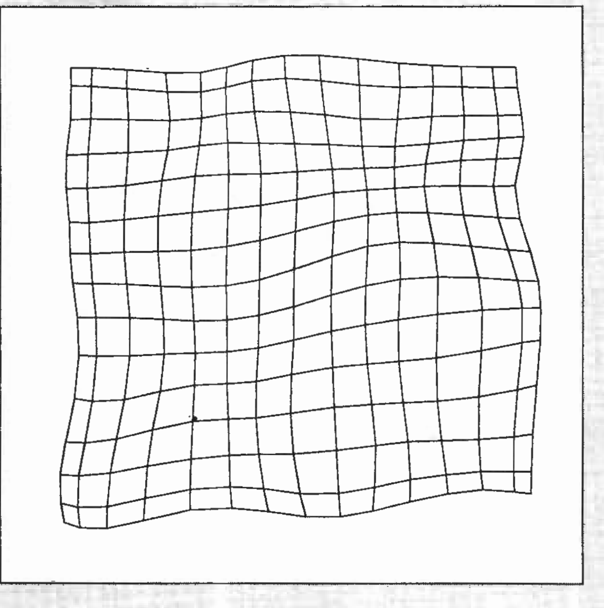


Bild 4. In den drei Abbildungen erkennt man deutlich, wie sich die Zuordnung der Rezeptoren zu den „Neuronen" entwickelt





Die Funktion H legt die Größe der Korrekturen und den Durchmesser der von ihnen erfaßten Nachbarschaftsregion um Neuron (i_0,j_0) fest.
Die Schritte 1. und 2. zusammen stellen jeweils einen Lernschritt dar und werden in einer Laufschleife 'maxtime' mal wiederholt, wobei der Sinnesreiz vor jedem Teilschritt durch Auswahl einer Stelle (x,y) mittels Zufallszahlengenerator nachgebildet wird (Prozedur 'stimulus'). Zusätzlich wird alle 20 Zeitschritte das bis dahin erreichte Stadium der Zuordnung zwischen den Neuronen und der Sinnesoberfläche graphisch dargestellt.
Bild 4 zeigt drei Entwicklungsstadien aus einem Programmlauf mit 'maxtime' (= 1000) Zeitschritten. Oben im Bild der Anfangszustand: die Zuordnung zwischen den Neuronen des Modellrindenfelds und den Punkten der Sinnesoberfläche ist völlig willkürlich und den meisten benachbarten Neuronen entsprechen weit voneinander entfernte Punkte der Sinnesoberfläche. Nach nur 20 Zeitschritten wird bereits eine Grobordnung erkennbar (Mitte), aber erst nach insgesamt 1000 Zeitschritten erhält man ein Ergebnis wie im unteren Bild: die Neuronen haben nun ihre Verbindungen in einer gitterförmigen Anordnung so über die Sinnesoberfläche ausgebreitet, daß benachbarte Neuronen wieder mit benachbart liegenden Rezeptoren verschaltet sind.
Am Schluß noch einige Hinweise für alle, die mit dem Programm (*Bild* 5) weiterexperimentieren möchten. Die Parameter H_0 und B_0 bestimmen die anfängliche Größe der Lernschritte und den Radius (in Gittereinheiten) der Region, innerhalb der nennenswerte Verschiebungen der $\vec{w}_{ij}$ erfolgen. Je größer H_0 und B_0 sind, desto rascher vollzieht sich die Grobordnungsphase. Jedoch muß aus Stabilitätsgründen
$0 < H_0 < 1$ sein. Beide Größen werden im Laufe des Selbstorganisationsprozesses auf Endwerte H_1 und B_1 exponentiell mit der Zeit verkleinert. Das Endresultat wird umso besser, je kleiner H_1 und B_1 gewählt werden, zugleich nimmt aber die Anzahl der erforderlichen Lernschritte zu, wenn der Ordnungsprozess nicht vorzeitig „einfrieren“ soll.
Neben Parameteränderungen kann man durch Modifizieren der Prozedur 'stimulus' auch die Form der Sinnesoberfläche und die Häufigkeitsverteilung der „Tastreize“ verändern. Auf diese Weise kann man z. B. demonstrieren, daß der beschriebene Selbstorganisationsprozess Regionen mit besonders häufigen Sinnesreizen mehr Neuronen zuordnet als seltener gereizten Gebieten, bzw. Regionen ohne Sinnesreizungen Neuronen entzieht.
Eine derartige durch die Reizfrequenz gesteuerte Plastizität ist für verschiedene Projektionen zwischen Nervenzellen höherer Tiere beobachtet worden. Die Nachbildung dieser Fähigkeit im Computer erlaubt einem Programm, oft benötigte Daten automatisch detaillierter zu speichern.
Diese Eigenschaft ist z. B. für die Bewegungssteuerung von Robotern sehr attraktiv: die verfügbaren „Neuronen“ verteilen sich im Laufe der Zeit entsprechend der Ausführungshäufigkeit über die einzelnen Bewegungen. Dabei entspricht der Aufbau der Verbindungen zwischen den Schichten einem Lernvorgang. In einer Anwendung haben wir auf diese Weise einem Modellroboter das Balancieren eines Stabes „beigebracht“, indem der Roboter durch Üben mit einem Stab sich selbst programmierte.

Assoziatives Gedächtnis

Von den vielen intelligenten Fähigkeiten unseres Gehirns beeindruckt das Assoziieren besonders. Wird uns Menschen eine schon einmal erlebte Situation kurz geschildert, dann reichen zum Teil schon einzelne Stichworte aus, um sehr lebendige, vielschichtige Eindrücke wachzurufen. Unser Gedächtnis läßt vor unserem geistigen Auge ganze Serien von Bildern erscheinen, sobald wir an prägende Ereignisse erinnert werden.

```
program Selbstorganisierende_Abbildung(input,output); (* H. Ritter, Juli 87 *)

const ni=15; nj=15;        (* Kantenlaengen des Neuronen-Gitters *)
      npts = 4;            (* Anzahl Eckpunkte der Sinnesoberflaeche. *)

type  vector   =  array [1..2] of real;
      neurons  =  array [1..ni,1..nj] of vector;

var   w                                : neurons;
      xy                               : vector;
      border                           : array [0..npts] of vector;
      ch                               : char;
      a0,b0,c0,h0,b1,h1                : real;
      i0,j0,time,maxtime,seed          : integer;

procedure line1(u,v: vector);  (* Zeichnet Verbindungslinie zwischen u und v *)
   var m1,m2,n1,n2: integer;
begin  (* Abbildung des Quadrats [0,1]*[0,1] auf 300x300-Pixel-Schirmregion: *)
   m1:=170 + round(300*u[1]);  n1:=330 - round(300*u[2]);
   m2:=170 + round(300*v[1]);  n2:=330 - round(300*v[2]);
   line(m1,n1,m2,n2,1,0,0,0,-1,1);
end;

procedure clear;               (* Loescht Mitteltteil des Bildschirms *)
   var pat,patmask: integer;
begin
   pat:=0; patmask:=0;
   fill_rectangle(0,0,639,367,1,0,0,0,0,pat,patmask)
end;

function ran(var i:integer): real;  (* Jeder Aufruf liefert als  Funktions- *)
begin                              (* wert eine Pseudozufallszahl  zwischen *)
   i:=(10007*i+1);                 (* 0 und 1. I ist Hilfsvariable  und nur *)
   ran:= 0.5*(1+i/maxint);         (* beim ersten Aufruf mit einem belie-   *)
end;                               (* bigen, die Pseudozufallszahlenfolge   *)
                                   (* bestimmenden Anfangswert zu belegen.  *)

procedure bestmatch(var i0: integer; var j0: integer; xy  : vector);
(* Sucht Neuron (I0,J0), dessen Verbindungsstelle die kuerzeste Distanz
   zur Reizstelle XY aufweist. *)
   var i,j: integer;      s1,s2,dist,newdist: real;
begin
   dist:=1E20;                    (* pessimistischer Startwert fuer Distanz *)
   s1:=xy[1]; s2:=xy[2];
   for i:=1 to ni do         (* fuer alle Neuronen nachsehen, ob unterboten *)
      for j:=1 to nj do begin
         newdist:=abs(s1-w[i,j,1])+abs(s2-w[i,j,2]);
         if newdist < dist then begin           (* falls ja, dist ersetzen,*)
            dist := newdist;                    (* und (i,j)-indexpaar als *)
            i0   := i;  j0 := j                 (* neues (i0,j0) nehmen.   *)
         end
      end
end;
```

Bild 5. Dieses Programm realisiert die „selbstorganisierende Abbildung“ in Bild 4 ▶

```
procedure adjust(i0,j0: integer; xy: vector; var w: neurons);
(* Fuehrt Adaptationsschritt fuer alle Verbindungen von Neuronen
   in der Umgebung von Neuron (i0,j0) aus. *)
   var  k,i,i1,i2,i3,j,j1,j2,j3,radius : integer;     b,h : real;
begin
   b := b0*exp(-c0*time);                             (* vgl. Text *)
   radius := trunc(2.0*b+1.0);
   if i0 > radius then i1:= i0-radius else i1:= 1;    (* Gitterbereich  um *)
   i2:=i0+radius ;      if i2 > ni then i2:=ni;       (* (i0,j0) abstecken *)
   if j0 > radius then j1:= j0-radius else j1:= 1;
   j2:=j0+radius ;      if j2 > nj then j2:=nj;
   for i:=i1 to i2 do
      for j:=j1 to j2 do begin    (* alle Verbindungen w[i,j]  mit i1<i<i2 *)
         i3:=i0-i;  j3:=j0-j;     (* und j1<j<j2 nach w[i0,j0] verschieben.*)
         h := h0 * exp( - a0*time - (i3*i3+j3*j3)/(b*b));
         for k:=1 to 2 do   w[i,j,k] := w[i,j,k]*(1-h) + h*xy[k];
      end;
end;

procedure makexy(var xy: vector);       (* Waehlt Zufalls-Stimulus xy=:(x,y) *)
begin
   xy[1]:=ran(seed);
   xy[2]:=ran(seed);
end;

procedure initialize;               (* Erzeugt zufaellige Anfangsverschaltung *)
   var i,j,k : integer;
begin
   for i:=1 to ni do
   for j:=1 to nj do
   for k:=1 to 2 do
      w[i,j,k]:= ran(seed);
end;

procedure draw;
(* Zeichnet Umrandung und die mit den Gitter-Neuronen verbundenen
   Punkte der Sinnesoberflaeche. *)
   var i,j: integer;
begin
   for i:=0 to npts-1  do line1(border[i],border[i+1]);          (* Umrandung *)
   for i:=1 to ni do
      for j:=1 to nj-1 do line1(w[i,j],w[i,j+1]);             (* Verbindungen *)
   for j:=1 to nj do
      for i:=1 to ni-1 do line1(w[i,j],w[i+1,j]);
end;

procedure setparameters;                      (* Initialisiert Parameterwerte *)
begin   seed := 7771;           (* Initialisierung Zufallszahlengenerator *)
   (* Parameter fuer den Selbstorganisationsprozess: *)
   h0:=0.9;   h1:=0.02;
   b0:=5.0;   b1:=1.0;
   a0:=ln(h0/h1)/maxtime;           (* Abnahmerate Verschiebungsamplitude *)
   c0:=ln(b0/b1)/maxtime;           (* Abnahmerate Radius Korrekturgebiet *)
   (* Koordinatewerte fuer die Ecken einer quadratischen Sinnesoberflaeche:*)
   border[0,1]:= 0;       border[0,2]:= 0;
   border[1,1]:= 1;       border[1,2]:= 0;
   border[2,1]:= 1;       border[2,2]:= 1;
   border[3,1]:= 0;       border[3,2]:= 1;
   border[4,1]:= 0;       border[4,2]:= 0;
end;

(*  Hauptprogramm : *)

begin        (* Eingabe und Initialisierung: *)
   write('      Wieviele Zeitschritte? :      ');    read(maxtime);
   setparameters;
   initialize;
   time:=0;
   repeat    (* Lernschritte: *)
      if (time=10) or (time mod 20 = 0) then begin          (* Ausgabe *)
         writeln('                              Zeitschritte:  ',time);
         clear;  draw;
      end;
      makexy(xy);                   (* Erzeuge "Stimulus" *)
      bestmatch(i0,j0,xy);          (* Finde angesprochenes Neuron     *)
      adjust(i0,j0,xy,w);           (* Adaptationsschritt in Umgebung *)
      time := time +1;                                (* von Neuron i0,j0 *)
   until time=maxtime;
   writeln('                         Zeitschritte:  ',time);
   clear; draw;                   (* Abschliessende Ausgabe *)
   write('                         Beenden = beliebige Taste. ');
   read(ch);
end.
```

Ebenso stellen wir uns gewöhnlich etwas Konkretes vor, wenn wir von Bäumen, Häusern oder Personen reden. Wie wird die Information, die in den verschiedenen assoziativen Rindenfeldern unseres Gehirns gespeichert ist, gelernt und damit in die Nervennetzwerke eingeprägt? Wie wird sie bei Bedarf ausgelesen?

Eine Antwort auf diese Fragen hängt mit der besonderen Fähigkeit des biologischen Gedächtnisses zusammen, durch Assoziation, d. h. durch Angabe von Bruchstücken der gespeicherten Information die Gesamtinformation wachzurufen. Vor einigen Jahren sind von Physikern einfache Modellsysteme entdeckt worden, die mit ungeordneten magnetischen Materialien eng verwandt sind und diese Fähigkeiten ebenfalls zeigen. Im folgenden soll das vom amerikanischen Physiker Hopfield vorgeschlagene Modell eines assoziativen Netzwerkes erläutert werden.

Das Hopfield'sche Netzwerk besteht aus einem ebenen, rechteckigen Gitter, an dessen Knotenpunkten Zeiger befestigt sind. Diese Zeiger können nur zwei Stellungen einnehmen, und werden durch Boole'sche Variablen $s[i] = 0,1$ beschrieben. Der Index $i = 1, 2, ..., n$ numeriert die Zeiger durch, wobei das Gitter $n = n_x \times n_y$ Gitterpunkte enthält und n_x und n_y die Zahl der Gitterknoten in x- und y-Richtung angeben. Der Zusammenhang zwischen dem Netzwerk aus Zeigern und einem physiologischen Nervennetzwerk besteht darin, daß Nervenzellen sehr vereinfacht durch zwei Zustände, den Aktivitätszustand und den Ruhezustand beschrieben werden können. Nützlich für ein Verständnis des folgenden ist die Vorstellung, daß die Zeiger im Netzwerk ein Muster anzeigen. Es sollen nur weiße und schwarze Bildpunkte in den Mustern zugelassen sein. Stellen Sie sich weiter vor, im Netzwerk seien p Pixelmuster gespeichert, die durch Zahlen
$s_i^\nu = 0, 1; i = 1, 2, ..., n; \nu = 1, 2, ..., p$
dargestellt werden. Wählen wir nun im Hopfield'schen Netzwerk irgendwelche Zeigerstellungen als Anfangszustand des Netzwerkes, dann führen die von Hopfield postulierten dynamischen Regeln für Zitterbewegungen der Zeiger dazu, daß die Zeiger nach einiger Zeit ein gespeichertes Muster einstellen. Das Netzwerk sucht dabei dasjenige Muster heraus, das der anfänglichen Ausrichtung am nächsten kommt. Mit Hilfe der Dynamik assoziiert das Netzwerk zu einer bruchstückhaften Eingabe eines Musters ein eingespeichertes, korrektes Muster. Wir stoßen in unserem Modell-

```
program Assoziatives_Gedaechtnis (input,output);
(* Juli 1987; Joachim Buhmann *)
const nx=10; ny=10; (* Laenge nx und Breite ny des Neuronennetzes *)
      n = 100;      (* n=nx*ny Zahl der Neuronen *)
      pmax=9;       (* pmax<=n*0.14 Maximale Speicherkapazitaet *)
var i, it, itout,   (* Hilfsvariablen *)
    lrs, lsp,       (* Speicherplatz fuer Zufallszahl *)
    irs,            (* Nummer des zufaellig ausgewaehlten Neurons *)
    p,              (* Zahl der Muster *)
    idelta : integer; (* 1000*1/n: Normierung der Synapsenstaerke *)
    temp,           (* Staerke des Zufallspotentials *)
    e0,             (* Schwellpotential *)
    act,            (* Erregung des zufaellig ausgewaehlten Neurons irs*)
    delta : real;   (* 1/n *)
    s : array [1..n] of boolean;          (* aktueller Neuronenzustand *)
    sm : array [1..n,1..pmax] of boolean;  (* Musterzustaende *)
    w : array [1..n,1..n] of integer;      (* Synapsenmatrix *)

function ran (var i : integer) : real; (* Zufallszahlengenerator *)
   begin
   i := (6907 * i + 1);
   ran := 0.5*(1.0 - i / maxint );
   end;

function hamming (i : integer) : real;
(* Hamming-Abstand (Zahl der unterschiedlichen Bits) zwischen dem
   Netzwerkzustand und dem Muster i. *)
   var j : integer;
       ham : real;
   begin
   ham := 0;
   for j := 1 to n do
      if s[j] <> sm[j,i] then ham := ham + delta;
   hamming := ham;
   end;

function spikeprob (act : real) : real;
(* Wahrscheinlichkeit, mit der das Neuron irs zum Zeitpunkt t+1 feuert. *)
   var x : real;
   begin
   x := - act + e0;
   if abs(x) < temp*85 then spikeprob := 1.0/(1.0+exp(x/temp))
                       else spikeprob := ord (x < 0.0);
   end;

procedure patterns;
(* patterns erzeugt das Muster A und p-1 Zufallsmuster *)
   var im, i, ix, iy : integer;
       noise : real;      (* Hamming-Distanz zwischen dem Anfangszustand
                             und dem Muster A *)
       a : array[1..ny] of packed array[1..nx] of char;
   begin
      a[1] := '    **    ';
      a[2] := '    **    ';
      a[3] := '   ****   ';
      a[4] := '   ****   ';
      a[5] := '  **  **  ';
      a[6] := '  **  **  ';
      a[7] := ' ******** ';
      a[8] := ' ******** ';
      a[9] := '**      **';
      a[10] :='**      **';
      write ('Prozentsatz falscher Pixel in A: noise := ');
      readln (noise); writeln;
      for iy := 1 to ny do
         for ix := 1 to nx do
            begin
            i := (iy-1)*nx+ix;
            sm[i,1] := (a[iy,ix] = '*');
            if ran(lrs) > noise then s[i] := sm[i,1]
                                else s[i] := not sm[i,1];
            for im := 2 to p do sm[i,im] := (ran(lrs) > 0.5);
            end;
   end;

procedure output_sp;
(* Ausgabeprozedur fuer den Netzwerkzustand s[0..n] *)
   var ix, iy, i : integer;
   begin
      writeln ('t = ',it:4,' Iterationen');
      for ix := 0 to nx+1 do write('-'); writeln;
      for iy := 1 to ny do
         begin
         write ('|');
         for ix := 1 to nx do
            if s[(iy-1)*nx+ix] then write('*')
                               else write(' ');
         write ('|');
         if iy <= p then write ('     hd',iy:1,' = ',hamming(iy):3:2);
         writeln;
         end;
      for ix := 0 to nx+1 do write('-'); writeln; writeln;
    end;

procedure store; (* store besetzt die Synapsenstaerken w[i,k] zwischen
den Neuronen i und k *)
   var i, k, im : integer;
   begin
      for k := 1 to n do
         for i := k+1 to n do
            begin
            w[i,k] := 0;
            for im := 1 to p do
               w[i,k] := w[i,k] + idelta*(2*ord(sm[i,im]=sm[k,im])-1);
               w[k,i] := w[i,k];
            end;
   end;

begin
   lsp := 9601; lrs := 5819;
   delta := 1.0/n; idelta := round(1000/n);
   itout := n;
   write ('Temperatur temp := '); readln (temp); writeln;
   write ('Schwellpotential e0 := '); readln (e0); writeln;
   write ('Zahl der Muster p := '); readln (p); writeln;
   patterns;
   store;
   for it := 0 to 100*n do
      begin                (* Beginn eines Monte Carlo Schritt *)
      if (it mod itout) = 0 then output_sp;
      irs := trunc(ran(lrs)*(n-1)+1.0);
      act := 0.0;
      for i := 1 to n do if s[i] then act := act + w[i,irs]/1000;
      s[irs] := (ran(lsp) <= spikeprob(act));
      end;
end.
```

Bild 6. Mit einem Hopfield-Netzwerk wird in diesem Programm assoziatives Erinnerungsvermögen simuliert

netzwerk auf eine Fähigkeit, die unser eigenes Gedächtnis auch aufweist.
Die dynamischen Regeln des assoziativen Netzwerkes beruhen auf Wechselwirkungen w_{ik} zwischen Zeigern i und k. Hopfield führte Wechselwirkungen ein, die mit den Regeln

Formel 3:

$$w_{ik} = \frac{1}{n} \sum_{v} (2s_i^v - 1)\,(2s_k^v - 1)$$

bestimmt werden und auf den Psychologen Hebb zurückgehen. Die Variablen w_{ik} enthalten offensichtlich die Information über die zu speichernden Muster s_i^v. Wenn die oben genannten Regeln verwendet werden, sollten die Muster allerdings aus etwa gleich vielen schwarzen wie weißen Bildpunkten zusammengesetzt sein.

Die Zeiger werden nun einer Zitterbewegung ausgesetzt, das heißt es wird versucht, einen zufällig ausgewählten Zeiger i umzuorientieren, also die zugeordneten Variablen s_i zu ändern. Dabei hält man sich an folgende Vorschrift: Man berechnet zunächst für den i-ten Zeiger die Größe

Formel 4:

$$f = \left[1 + \exp\left[- \frac{\sum_k w_{ik} s_k - e_0}{temp} \right] \right]^{-1}$$

deren Wert im Bereich [0, 1] liegt. Danach wird eine Funktion ('function ran') aufgerufen, die eine im Intervall [0,1] gleichverteilte Zufallszahl liefert, z. B. die Zahl r. Falls r kleiner als f ist, wird dem i-ten Zeiger der Wert 1 zugewiesen, im umgekehrten Fall der Wert 0. Diese Prozedur wird dann auf den nächsten, zufällig herausgegriffenen Zeiger angewendet usw. Im obigen Ausdruck für f stellt e_0 eine Schwelle dar, die normalerweise gleich Null gesetzt werden kann. Werden im Netzwerk aber Muster gespeichert, die nicht mehr aus gleichvielen schwarzen wie weißen Bildpunkten bestehen, dann muß der Parameter e_0 neu eingestellt werden. Wie stark die Zitterbewegung eines Zeiger ist, wird mit dem Parameter 'temp' gemessen. Sehr große Werte für 'temp' bewirken, daß $f \simeq 0{,}5$ ist und beide Stellungen des herausgegriffenen Zeigers etwa gleich häufig auftreten. In diesem Fall werden die Zeiger schnell durch ihre Zitterbewegung ungeordnet und das System ist nicht fähig, ein bestimmtes Muster zu assoziieren. Bei sehr kleinen Werten für 'temp' sind nur Zeigerbewegungen möglich, bei denen die Summe $\Sigma\, w_{ik}\, s_k$ größer als die Schwelle e_0 ist. Dies bedeutet bei größeren Netzwerken, daß die Zeiger sehr schnell in einem Muster gefangen bleiben, welches nicht unbedingt einem gespeicherten Muster entspricht. Bei Temperaturen temp < 1, die aber nicht zu klein sein dürfen, hat das Zeigersystem noch eine genügende Chance, durch Zitterbewegung korrekte Muster zu erreichen und wird nur sehr selten nach Erreichen eines gespeicherten Musters in neue Orientierungen getrieben.

Wie genau das Netzwerk ein gespeichertes Muster wiedergibt, wird durch den Prozentsatz der Zeiger bestimmt, die nicht mit dem Muster übereinstimmen ('function hamming').

Das Hopfield-Netzwerk, das im beigefügten Programm (*Bild 6*) simuliert wird, hat p Muster gespeichert. Als erstes Muster wird aus Gründen der leichten Erkennbarkeit der Buchstabe A gewählt, die übrigen Muster sind gewürfelt und besitzen gleich viele schwarze wie weiße Bildpunkte. Der Leser kann hier auch eigene Muster wählen, sollte aber darauf achten, daß die Muster aus ungefähr gleich vielen schwarzen wie weißen Bildpunkten besteht.

Als Anfangsstellung der Zeiger wählen wir ein verrauschtes A. Dazu werden die Zeiger in die gleiche Stellung wie im Muster A gebracht und zusätzlich werden einige Zeiger umgedreht.

In *Bild 7* ist ein typischer Simulationslauf mit den Parametern temp = 0,0; e_0 = 0,0; p = 5 dargestellt. Der Anfangszustand des Netzwerkes unterscheidet sich in 19% der Bildpunkte vom Muster A. Innerhalb von typischerweise 400...500 Iterationen verbessert das Netzwerk alle Fehler und erinnert sich an das gespeicherte Muster A.

Die aktuelle Forschung bemüht sich zur Zeit, verschiedene technische und biologische Fragestellungen mit assoziativen Speichern zu lösen. So wird etwa ein Chip zur Kompression der Datenmengen, die beim Bildtelefon anfallen, entwickelt. Andere Arbeitsgruppen versuchen, den Geruchssinn zu erklären, d. h. die Assoziation der Anregung vieler Geruchsrezeptoren mit dem Geruch eines bestimmten Stoffes. In unserem Institut studieren wir assoziative Speicher für Musterfolgen und zur Lösung des Invarianzproblemes beim Erkennen von Bildern, d. h. assoziative Speicher mit der Fähigkeit, Gegenstände ohne Rücksicht auf ihre Lage, Orientierung und Größe zu erkennen.

```
Temperatur       temp := 0.0
Schwellpotential e0 := 0.0
Zahl der Muster    p := 5
Prozentsatz falscher Pixel
in A: noise := 0.2


t =    0 Iterationen
------------
|     *   *|      hd1 = 0.19
|*   ***   |      hd2 = 0.57
|  *****   |      hd3 = 0.50
|   **** * |      hd4 = 0.56
|  *       |      hd5 = 0.55
|  **   * *|
| ** ****  |
| ***** ** |
|**  *   * |
|         *|
------------

t =  100 Iterationen
------------
|   ***    |      hd1 = 0.10
|*   **    |      hd2 = 0.54
|  * ***   |      hd3 = 0.47
|   **** * |      hd4 = 0.55
|  **   *  |      hd5 = 0.56
|  **  ** *|
| ** ****  |
| ******** |
|**      **|
| *      **|
------------

t =  400 Iterationen
------------
|    **    |      hd1 = 0.00
|    **    |      hd2 = 0.54
|   ****   |      hd3 = 0.49
|   ****   |      hd4 = 0.51
|  **  **  |      hd5 = 0.52
|  **  **  |
| ******** |
| ******** |
|**      **|
|**      **|
------------
```

Bild 7. Ein typischer Simulationslauf des Programmes aus Bild 6: Der Buchstabe 'A' wird vom Netzwerk erkannt

Literatur

[1] *Schulten, K.*: Ordnung aus Chaos – Vernunft aus Zufall, in *Küppers, B. O.* (Herausgeber): Ordnung aus dem Chaos, Piper-Verlag, Mai 1987.

Der Compiler kann den erzeugten Code wahlweise hinsichtlich einer effizienten Speicherausnutzung (Size) oder hinsichtlich einer maximalen Geschwindigkeit (Speed) optimieren: Im einen Fall erfolgen Aufrufe von Routinen im Laufzeit-Paket mit Software-Interrupts, im anderen Fall mit „far calls", d. h. mit Intersegment-Sprüngen.

Wie der Bildschirm nach dem Compilieren des Testprogramms aussehen kann, zeigt *Bild 2*: Der Cursor steht bei einem ON-ERROR-Statement, das zu einer Fehlermeldung „Missing On Error option" führte.

Die bei Version 1.0 noch vorhandene Option /a, die es ermöglichte, den erzeugten Code als Disassembler-Listing anzuzeigen, fiel seit Version 2.0 leider weg. Das ist ein wenig schade, weil dem Programmierer dadurch das Verständnis entzogen wird, wie man einen möglichst effizienten und schnellen Objektcode erzeugen kann.

Übrigens ist es bei Version 3.0 nach wie vor möglich, den Compiler aus einer Batch-Datei heraus zu starten; er meldet sich dann nicht mit dem Editor, sondern holt sich das Quellprogramm und erzeugt daraus ein Objekt-File für den Linker.

Die Geschwindigkeit

Bei einem Compiler gibt es zwei relevante Geschwindigkeits-Angaben: zum einen, wie schnell die Übersetzung eines Quellprogramms in ein Objektprogramm abläuft, zum anderen, wie schnell das erzeugte Objektprogramm selbst ist.

Microsoft gibt an, Quickbasic könne „bis zu 12000 Zeilen pro Minute" übersetzen. Bei unserem Test wurde ein etwa 300 Zeilen langes Basic-Programm (mit Zeilennummern, wie der Interpreter es liefert, und mit durchschnittlich 60 Zeichen je Zeile) auf einem 8-MHz-AT in rund 11 Sekunden übersetzt. Dabei wurde die Option „output to memory" verwendet, die verständlicherweise am schnellsten arbeitet. Diese Geschwindigkeit entspricht etwa 1600 Zeilen je Minute – erreicht also den von Microsoft genannten Idealwert bei weitem nicht, der vermutlich nur dann gilt, wenn je Zeile nur ein Befehl übersetzt werden muß und ein 80386-Rechner zum Einsatz kommt.

Der in mc schon öfter verwendete Mini-Benchmark-Test (2000 Divisionen 10/3 mit Variablen einfacher Genauigkeit) wurde, um zu meßbaren Zeiten zu kommen, auf 20 000 Schleifendurchläufe „aufgeblasen". Er benötigte dann für das siebenstellige Ergebnis mit dem GW-Basic-Interpreter auf einem 8-MHz-AT rund zwanzig Sekunden und mit dem Quickbasic-Compiler 4,3 s:

```
10 T=TIMER
20 FOR I=1 TO 20000
30 A=10/3
40 NEXT I
50 PRINT A, TIMER-T
```

Ersetzt man die Real-Variable I als Schleifenzähler durch eine Integer-Variable, so wird der Unterschied krasser: 19 s beim Interpreter und 1,9 s beim Compiler. Obwohl also A immer noch siebenstellig berechnet wird, ist der Compiler hier zehnmal schneller. Allgemein läßt sich sagen, daß der Geschwindigkeitsvorteil des Compilers umso größer wird, je mehr man von Integer-Variablen Gebrauch macht, da 16-Bit-Operationen direkt von der CPU ausgeführt werden können und keine zeitraubenden Fließkomma-Umformungen nötig sind. Durch konsequente Anwendung dieses Grundsatzes ist es oftmals möglich, compilierte Programme zwanzigmal schneller als interpretierte zu machen.

Neues von MS-OS/2

OS/2 ist ein Multitasking-Betriebssystem das die 640 KByte-Grenze von MS-DOS durchbricht. Es unterstützt den gesamten Adreßraum des 80286-Prozessors, d.h. 16 MByte RAM und 1 GByte virtueller Speicher.
Die Schnittstelle zu Anwenderprogrammen, API (application programm interface) genannt, beruht auf CALL-Befehlen. Der MS-OS/2 Windows Presentation Manager verleiht dem neuen Betriebsystem eine grafische Bedieneroberfläche.
Laut Microsoft wird das neue Betriebssystem MS-OS/2 kompatibel mit allen MS-DOS-Versionen sein. Die meisten MS-DOS-Anwendungen werden unverändert unter dem neuen Betriebssystem laufen. Während einer Pressekonferenz zeigte Microsoft, daß MS-Word, das teilweise direkt in den Bildschirmspeicher schreibt, unter OS/2 läuft.

Microsoft empfiehlt für MS-OS/2 ein System mit mindestens 1,5 MByte RAM und einer Festplatte. Selbstverständlich setzt der Windows Presention Manager eine Grafikkarte im System voraus. Er besteht aus einer umfangreichen Grafik-Bibliothek, einem geräteunabhängigen Ausgabe-Treiber und Schnittstellen zur Intertask-Kommunikation und zum Austausch von Daten zwischen verschiedenen Anwendungen. Laut Microsoft lassen sich Programme, die für Windows unter MS-DOS geschrieben wurden, leicht modifizieren, so daß diese Programme in der Multitasking-Umgebung von OS/2 laufen.
Das Standard-MS-OS/2 besteht aus dem Betriebssystemkern und dem Windows Presentation Manager. Die erweiterte Version wird durch den LAN-Manager ergänzt. Dieser ist in das Betriebssystem integriert. Damit kann die Intertask-Kommunikation über das gesamte Netzwerk stattfinden. Zur Programmausführung kann der Anwender Anwendungen auf andere PCs innerhalb des Netzwerkes exportieren. Vorausichtlich wird der LAN-Manager im ersten Quartal 1988 an OEM-Kunden ausgeliefert werden.

Nach Unterlagen von Microsoft

mc-EPROM-Brenner

In dem Beitrag „mc-EPROM-Brenner" (Heft 1/1987) haben sich zwei Fehler eingeschlichen. Im Layout sind verschiedene Leitungen ein kurzes Stück zu dünn. Betroffen sind die Leitungen zum Pin 28 der EPROM-Fassung, zum Pin 14 von IC 8, zum Kollektor von T 12 und die Masseleitung des Gleichrichters Gl. In der Prozedur LOAD_CMD (Seite 56, links oben) steht eine END-Anweisung zuwenig. Die ELSE-Klausel ist überflüssig.

St

```
 7: begin
     if (pos (PARAM[4], 'BIS ) = 1) and (pos (PARAM[6], 'NACH') = 1) then
     begin H2 := DECIMAL(PARAM[5], E2); H3 := DECIMAL_CHECK(PARAM[7], E3) end;
    end;
end;
 if E1 + E2 + E3 = 0 then
```

Der korrigierte Programmabschnitt